ISO/IEC JTC 1/SC 22

Programming languages, their environments and system software interfaces

Secretariat: ANSI

Document type:	Text for FCD ballot or comment
Title:	ISO/IEC FCD 1989, Information technology - Programming Languages, their environments and system software interfaces - Programming Language COBOL
Status:	The disposition of comments on the CD ballot have been distributed as SC 22 N 4562. ISO/IEC CD 1989 was distributed as SC 22 N 4315 and the Summary of Voting as SC 22 N 4357. Please submit all votes and comments via the Balloting Portal.
Date of document:	2010-08-06
Source:	SC 22 WG 4 Convenor (W. Takagi)
Expected action:	VOTE
Action due date:	2010-12-07
No. of pages:	901
Email of secretary:	mpeacock@ansi.org
Committee URL:	http://isotc.iso.org/livelink/livelink/open/jtc1sc22

ISO/IEC/JTC 1/SC 22/WG 4 **N 0xxx**

Date: 2010-07-13

Reference number of document: **ISO/IEC 1989:20xx FCD 1.0**

Information technology — Programming languages, their environments and system software interfaces —

Programming language COBOL

Document type: Final Committee Draft International Standard
Document subtype: Not applicable
Document stage: (4) Final Committee Draft International Standard
Document language: E

Contents

Contents ii

Tables xix

Figures xx

Foreword xxi

Introduction xxii

1 Scope 1

2 Normative references 2

3 Conformance to this final committee draft International Standard 3
3.1 General 3
3.2 A conforming implementation 3
3.2.1 Acceptance of standard language elements 3
3.2.2 Interaction with non-COBOL runtime modules 3
3.2.3 Interaction between COBOL implementations 3
3.2.4 Implementor-defined language elements 3
3.2.5 Processor-dependent language elements 4
3.2.6 Optional language elements 4
3.2.7 Reserved words 4
3.2.8 Standard extensions 4
3.2.9 Nonstandard extensions 4
3.2.10 Substitute or additional language elements 5
3.2.11 Archaic language elements 5
3.2.12 Obsolete language elements 5
3.2.13 Externally-provided functionality 5
3.2.14 Limits 6
3.2.15 User documentation 6
3.2.16 Character substitution 6
3.3 A conforming compilation group 6
3.4 A conforming run unit 6
3.5 Relationship of a conforming compilation group to a conforming implementation 7
3.6 Relationship of a conforming run unit to a conforming implementation 7

4 Terms and definitions 8

5 Description techniques 18
5.1 General 18
5.2 General formats 18

5.2.1 Keywords . . . 18
5.2.2 Optional words . . . 18
5.2.3 Operands . . . 18
5.2.4 Level numbers . . . 19
5.2.5 Options . . . 19
5.2.5.1 Brackets . . . 19
5.2.5.2 Braces . . . 19
5.2.5.3 Choice indicators . . . 19
5.2.6 Ellipses . . . 19
5.2.7 Punctuation . . . 20
5.2.8 Special characters . . . 20
5.2.9 Meta-terms . . . 20
5.3 Rules . . . 20
5.3.1 Syntax rules . . . 20
5.3.2 General rules . . . 20
5.3.3 Argument rules . . . 20
5.3.4 Returned value rules . . . 20
5.4 Arithmetic expressions . . . 21
5.4.1 Textually subscripted operands . . . 21
5.4.2 Ellipses . . . 21
5.5 Integer operands . . . 21
5.6 Informal description . . . 21
5.7 Hyphens in text . . . 22
5.8 Verbal forms for the expression of provisions . . . 22

6 Reference format . . . 23
6.1 General . . . 23
6.2 Indicators . . . 23
6.2.1 Fixed indicators . . . 23
6.2.2 Floating indicators . . . 24
6.3 Fixed-form reference format . . . 25
6.4 Free-form reference format . . . 27
6.5 Logical conversion . . . 28

7 Compiler directing facility . . . 30
7.1 General . . . 30
7.2 Text manipulation . . . 30
7.2.1 Text manipulation elements . . . 31
7.2.2 COPY statement . . . 33
7.2.3 REPLACE statement . . . 37
7.3 Compiler directives . . . 40
7.3.1 General format . . . 40
7.3.2 Syntax rules . . . 40
7.3.3 General rules . . . 40
7.3.4 Conditional compilation . . . 41
7.3.5 Compile-time arithmetic expressions . . . 41

7.3.6 Compile-time boolean expressions 41
7.3.7 Constant conditional expression 42
7.3.8 CALL-CONVENTION directive 44
7.3.9 DEFINE directive 45
7.3.10 EVALUATE directive 46
7.3.11 FLAG-02 directive 49
7.3.12 FLAG-85 directive 50
7.3.13 FLAG-NATIVE-ARITHMETIC directive 52
7.3.14 IF directive 53
7.3.15 LEAP-SECOND directive 54
7.3.16 LISTING directive 55
7.3.17 PAGE directive 56
7.3.18 PROPAGATE directive 57
7.3.19 SOURCE FORMAT directive 58
7.3.20 TURN directive 59

8 Language fundamentals 61
8.1 Character sets 61
8.1.1 Computer's coded character set 61
8.1.2 COBOL character repertoire 62
8.1.3 Alphabets 65
8.1.4 Collating sequences 65
8.2 Locales 66
8.2.1 Locale field names 67
8.3 Lexical elements 68
8.3.1 Character-strings 68
8.3.1.1 COBOL words 68
8.3.1.2 Literals 76
8.3.1.3 Picture character-strings 84
8.3.2 Separators 84
8.4 References 86
8.4.1 Uniqueness of reference 86
8.4.1.1 Qualification 86
8.4.1.2 Subscripts 88
8.4.2 Identifiers 90
8.4.2.1 Identifier 90
8.4.2.2 Function-identifier 92
8.4.2.3 Reference-modification 95
8.4.2.4 Inline method invocation 97
8.4.2.5 Object-view 98
8.4.2.6 EXCEPTION-OBJECT 99
8.4.2.7 NULL object reference 99
8.4.2.8 SELF and SUPER 99
8.4.2.9 Object property 100
8.4.2.10 NULL address pointer 101
8.4.2.11 Data-address-identifier 102

8.4.2.12 Function-address-identifier . . . 103
8.4.2.13 Program-address-identifier . . . 104
8.4.2.14 LINAGE-COUNTER . . . 104
8.4.2.15 Report counters . . . 105
8.4.3 Condition-name . . . 106
8.4.4 Explicit and implicit references . . . 106
8.4.5 Scope of names . . . 106
8.4.5.1 Local and global names . . . 107
8.4.5.2 Scope of program-names . . . 109
8.4.5.3 Scope of class-names and interface-names . . . 109
8.4.5.4 Scope of method-names . . . 109
8.4.5.5 Scope of function-prototype-names . . . 110
8.4.5.6 Scope of user-function-names . . . 110
8.4.5.7 Scope of program-prototype-names . . . 110
8.4.5.8 Scope of compilation-variable-names . . . 110
8.4.5.9 Scope of parameter-names . . . 110
8.4.5.10 Scope of property-names . . . 110
8.5 Data description and representation . . . 111
8.5.1 Computer independent data description . . . 111
8.5.1.1 Files and records . . . 111
8.5.1.2 Levels . . . 111
8.5.1.3 Limitations of character handling . . . 112
8.5.1.4 Algebraic signs . . . 112
8.5.1.5 Alignment of data items in storage . . . 113
8.5.1.5.1 Alignment of alphanumeric groups and of data items of usage display . . . 113
8.5.1.5.2 Alignment of data items of usage national . . . 113
8.5.1.5.3 Alignment of data items of usage bit . . . 113
8.5.1.5.4 Item alignment for increased object-code efficiency . . . 114
8.5.1.5.5 Alignment of strongly-typed group items . . . 114
8.5.1.6 Tables . . . 114
8.5.1.6.1 Fixed-capacity tables . . . 114
8.5.1.6.2 Occurs-depending tables . . . 115
8.5.1.6.3 Dynamic-capacity tables . . . 115
8.5.1.7 Any-length elementary items . . . 117
8.5.1.7.1 Structure of an any-length elementary item . . . 117
8.5.1.7.2 Location of any-length elementary items . . . 117
8.5.1.7.3 Operations on any-length elementary items . . . 117
8.5.1.8 Variable-length data items . . . 118
8.5.1.8.1 Contiguity of data items . . . 118
8.5.1.8.2 Availability and persistence of data items . . . 118
8.5.1.9 Variable-length groups . . . 118
8.5.1.9.1 Positional correspondence . . . 119
8.5.1.9.2 Matching . . . 119
8.5.2 Class and category of data items and literals . . . 119
8.5.2.1 Alphabetic category . . . 120
8.5.2.2 Alphanumeric category . . . 120

8.5.2.3 Alphanumeric-edited category 120
8.5.2.4 Boolean category 121
8.5.2.5 Data-pointer category 121
8.5.2.6 Function-pointer category 121
8.5.2.7 Index category 121
8.5.2.8 National category 121
8.5.2.9 National-edited category 121
8.5.2.10 Numeric category 122
8.5.2.11 Numeric-edited category 122
8.5.2.12 Object-reference category 122
8.5.2.13 Program-pointer category 122
8.5.3 Types 122
8.5.3.1 Weakly-typed items 123
8.5.3.2 Strongly-typed group items 123
8.5.4 Zero-length items 123
8.6 Scope and life cycle of data 124
8.6.1 Global names and local names 124
8.6.2 External and internal items 124
8.6.3 Automatic, initial, and static internal items 125
8.6.4 Based entries and based data items 126
8.6.5 Common, initial, and recursive attributes 126
8.6.6 Sharing data items 126
8.7 Operators 128
8.7.1 Arithmetic operators 128
8.7.2 Boolean operators 128
8.7.3 Concatenation operator 128
8.7.4 Invocation operator 128
8.7.5 Relational operators 128
8.7.6 Logical operators 130
8.8 Expressions 131
8.8.1 Arithmetic expressions 131
8.8.1.1 Native, standard, standard-binary, and standard-decimal arithmetic 131
8.8.1.2 Native arithmetic 132
8.8.1.3 Standard arithmetic 132
8.8.1.3.1 Standard intermediate data item 132
8.8.1.3.2 Addition 134
8.8.1.3.3 Subtraction 134
8.8.1.3.4 Multiplication 135
8.8.1.3.5 Division 135
8.8.1.3.6 Exponentiation 135
8.8.1.3.7 Unary plus 136
8.8.1.3.8 Unary minus 136
8.8.1.4 Standard-binary arithmetic 136
8.8.1.4.1 Standard-binary intermediate data item 136
8.8.1.4.2 Basic arithmetic operations in standard-binary arithmetic 136
8.8.1.4.3 Exponentiation in standard-binary arithmetic 137

8.8.1.5 Standard-decimal arithmetic . . . 138
8.8.1.5.1 Standard-decimal intermediate data item . . . 138
8.8.1.5.2 Basic arithmetic operations in standard-decimal arithmetic . . . 138
8.8.1.5.3 Exponentiation in standard-decimal arithmetic . . . 138
8.8.2 Boolean expressions . . . 139
8.8.3 Concatenation expressions . . . 142
8.8.4 Conditional expressions . . . 142
8.8.4.1 Simple conditions . . . 142
8.8.4.1.1 Relation conditions . . . 143
8.8.4.1.2 Boolean condition . . . 148
8.8.4.1.3 Class condition . . . 148
8.8.4.1.4 Condition-name condition (conditional variable) . . . 151
8.8.4.1.5 Switch-status condition . . . 151
8.8.4.1.6 Sign condition . . . 152
8.8.4.1.7 Omitted-argument condition . . . 152
8.8.4.2 Complex conditions . . . 153
8.8.4.2.1 Negated conditions . . . 153
8.8.4.2.2 Combined conditions . . . 153
8.8.4.2.4 Abbreviated combined relation conditions . . . 154
8.8.4.3 Order of evaluation of conditions . . . 155
8.9 Reserved words . . . 156
8.10 Context-sensitive words . . . 159
8.11 Intrinsic function names . . . 162
8.12 Compiler-directive words . . . 163
8.13 External repository . . . 164

9 I-O, objects, and user-defined functions . . . 165
9.1 Files . . . 165
9.1.1 Physical and logical files . . . 165
9.1.2 Record area . . . 165
9.1.3 File connector . . . 165
9.1.4 Open mode . . . 166
9.1.5 Sharing file connectors . . . 166
9.1.6 Fixed file attributes . . . 166
9.1.7 Organization . . . 166
9.1.7.1 Sequential . . . 167
9.1.7.2 Relative . . . 167
9.1.7.3 Indexed . . . 167
9.1.8 Access modes . . . 167
9.1.8.1 Sequential access mode . . . 168
9.1.8.2 Random access mode . . . 168
9.1.8.3 Dynamic access mode . . . 168
9.1.9 Reel and unit . . . 168
9.1.10 Current volume pointer . . . 168
9.1.11 File position indicator . . . 168
9.1.12 I-O status . . . 169

9.1.12.1 Successful completion 170
9.1.12.2 Implementor-defined successful completion 170
9.1.12.3 At end condition with unsuccessful completion 171
9.1.12.4 Invalid key condition with unsuccessful completion 171
9.1.12.5 Permanent error condition with unsuccessful completion 171
9.1.12.6 Logic error condition with unsuccessful completion 172
9.1.12.7 Record operation conflict condition with unsuccessful completion 173
9.1.12.8 File sharing conflict condition with unsuccessful completion 173
9.1.12.9 Implementor-defined condition with unsuccessful completion 173
9.1.13 Invalid key condition 173
9.1.14 Sharing mode 174
9.1.15 Record locking 175
9.1.16 Sort file 176
9.1.17 Merge file 176
9.1.18 Dynamic file assignment 176
9.1.19 Report file 176
9.2 Screens 177
9.2.1 Terminal screen 177
9.2.2 Function keys 177
9.2.3 CRT status 177
9.2.4 Cursor 178
9.2.5 Cursor locator 178
9.2.6 Current screen item 179
9.2.7 Color number 179
9.3 Objects 180
9.3.1 Objects and classes 180
9.3.2 Object references 180
9.3.3 Predefined object references 180
9.3.4 Methods 180
9.3.5 Polymorphism 180
9.3.5.1 Class polymorphism 180
9.3.5.2 Parametric polymorphism 180
9.3.6 Method invocation 181
9.3.7 Method prototypes 185
9.3.8 Conformance and interfaces 185
9.3.8.1 Conformance for object orientation 185
9.3.8.1.1 Interfaces 185
9.3.8.1.2 Conformance between interfaces 185
9.3.8.1.3 Conformance for parameterized classes and parameterized interfaces 187
9.3.9 Class inheritance 188
9.3.10 Interface inheritance 188
9.3.11 Interface implementation 188
9.3.12 Parameterized classes 188
9.3.13 Parameterized interfaces 189
9.3.14 Object life cycle 189
9.3.14.1 Life cycle for factory objects 189

9.3.14.2 Life cycle for instance objects . . . 189
9.4 User-defined functions . . . 189

10 Structured compilation group . . . 191
10.1 General . . . 191
10.2 Compilation units . . . 191
10.3 Source units . . . 191
10.4 Contained source units . . . 192
10.5 Source elements and runtime elements . . . 192
10.6 COBOL compilation group . . . 193
10.7 End markers . . . 198

11 Identification division . . . 199
11.1 General . . . 199
11.2 Identification division structure . . . 199
11.3 CLASS-ID paragraph . . . 200
11.4 FACTORY paragraph . . . 201
11.5 FUNCTION-ID paragraph . . . 202
11.6 INTERFACE-ID paragraph . . . 203
11.7 METHOD-ID paragraph . . . 204
11.8 OBJECT paragraph . . . 206
11.9 OPTIONS paragraph . . . 207
11.9.1 General format . . . 207
11.9.2 Syntax rules . . . 207
11.9.3 General Rules . . . 207
11.9.4 ARITHMETIC clause . . . 207
11.9.5 DEFAULT ROUNDED clause . . . 208
11.9.6 ENTRY-CONVENTION clause . . . 208
11.9.7 INTERMEDIATE ROUNDING clause . . . 209
11.10 PROGRAM-ID paragraph . . . 211

12 Environment division . . . 213
12.1 General . . . 213
12.2 Environment division structure . . . 213
12.3 Configuration section . . . 214
12.3.1 General format . . . 214
12.3.2 Syntax rules . . . 214
12.3.3 General rules . . . 214
12.3.4 SOURCE-COMPUTER paragraph . . . 215
12.3.5 OBJECT-COMPUTER paragraph . . . 216
12.3.6 SPECIAL-NAMES paragraph . . . 219
12.3.7 REPOSITORY paragraph . . . 230
12.4 Input-output section . . . 235
12.4.1 General format . . . 235
12.4.2 Syntax rules . . . 235
12.4.3 FILE-CONTROL paragraph . . . 236

12.4.4 File control entry 236
12.4.4.4 ACCESS MODE clause 242
12.4.4.5 ALTERNATE RECORD KEY clause 243
12.4.4.6 COLLATING SEQUENCE clause 244
12.4.4.7 FILE STATUS clause 246
12.4.4.8 LOCK MODE clause 247
12.4.4.9 ORGANIZATION clause 249
12.4.4.10 RECORD DELIMITER clause 250
12.4.4.11 RECORD KEY clause 251
12.4.4.12 RELATIVE KEY clause 252
12.4.4.13 RESERVE clause 253
12.4.4.14 SHARING clause 254
12.4.5 I-O-CONTROL paragraph 255
12.4.6 SAME clause 255

13 Data division 257
13.1 General 257
13.2 Data division structure 257
13.3 Explicit and implicit attributes 257
13.4 File section 258
13.4.1 General format 258
13.4.2 Syntax rules 258
13.4.3 General Rules 258
13.4.4 File description entry 259
13.4.5 Sort-merge file description entry 262
13.5 Working-storage section 263
13.6 Local-storage section 264
13.7 Linkage section 265
13.8 Report section 267
13.8.1 General format 267
13.8.2 Syntax rules 267
13.8.3 Report description entry 267
13.8.4 Report group description entry 267
13.8.5 Report subdivisions 267
13.8.5.1 Physical subdivisions of a report 267
13.8.5.1.1 Pages 267
13.8.5.1.2 Lines 268
13.8.5.1.3 Report Items 268
13.8.5.2 Logical Subdivisions of a Report 268
13.9 Screen section 269
13.10 Constant entry 270
13.11 Record description entry 272
13.12 Type declaration entry 272
13.13 77-level data description entry 272
13.14 Report description entry 272
13.15 Report group description entry 273

13.16 Data description entry . . . 276
13.17 Screen description entry . . . 281
13.18 Data division clauses . . . 285
13.18.1 ALIGNED clause . . . 285
13.18.2 ANY LENGTH clause . . . 286
13.18.3 AUTO clause . . . 288
13.18.4 BACKGROUND-COLOR clause . . . 289
13.18.5 BASED clause . . . 290
13.18.6 BELL clause . . . 291
13.18.7 BLANK clause . . . 292
13.18.8 BLANK WHEN ZERO clause . . . 293
13.18.9 BLINK clause . . . 294
13.18.10 BLOCK CONTAINS clause . . . 295
13.18.11 CLASS clause . . . 296
13.18.12 CODE clause . . . 297
13.18.13 CODE-SET clause . . . 298
13.18.14 COLUMN clause . . . 300
13.18.15 CONSTANT RECORD clause . . . 304
13.18.16 CONTROL clause . . . 305
13.18.17 DEFAULT clause . . . 307
13.18.18 DESTINATION clause . . . 308
13.18.19 Entry-name clause . . . 309
13.18.20 ERASE clause . . . 310
13.18.21 EXTERNAL clause . . . 311
13.18.22 FOREGROUND-COLOR clause . . . 312
13.18.23 FORMAT clause . . . 313
13.18.24 FROM clause . . . 315
13.18.25 FULL clause . . . 316
13.18.26 GLOBAL clause . . . 317
13.18.27 GROUP INDICATE clause . . . 318
13.18.28 GROUP-USAGE clause . . . 319
13.18.29 HIGHLIGHT clause . . . 320
13.18.30 INVALID clause . . . 321
13.18.31 JUSTIFIED clause . . . 322
13.18.32 Level-number . . . 323
13.18.33 LINAGE clause . . . 324
13.18.34 LINE clause . . . 326
13.18.35 LOWLIGHT clause . . . 330
13.18.36 NEXT GROUP clause . . . 331
13.18.37 OCCURS clause . . . 333
13.18.38 PAGE clause . . . 339
13.18.39 PICTURE clause . . . 341
13.18.40 PRESENT WHEN clause . . . 356
13.18.41 PROPERTY clause . . . 358
13.18.42 RECORD clause . . . 361
13.18.43 REDEFINES clause . . . 364

13.18.44 RENAMES clause 366
13.18.45 REPORT clause 367
13.18.46 REQUIRED clause 368
13.18.47 REVERSE-VIDEO clause 369
13.18.48 SAME AS clause 370
13.18.49 SECURE clause 372
13.18.50 SELECT WHEN clause 373
13.18.51 SIGN clause 374
13.18.52 SOURCE clause 375
13.18.53 SUM clause 377
13.18.54 SYNCHRONIZED clause 381
13.18.55 TO clause 383
13.18.56 TYPE clause 384
13.18.57 TYPEDEF clause 389
13.18.58 UNDERLINE clause 390
13.18.59 USAGE clause 391
13.18.60 USING clause 397
13.18.61 VALIDATE-STATUS clause 398
13.18.62 VALUE clause 400
13.18.63 VARYING clause 407

14 Procedure division 409
14.1 General 409
14.2 Procedure division structure 409
14.3 Declaratives 412
14.4 Procedures 413
14.4.1 Sections 413
14.4.2 Paragraphs 413
14.5 Procedural statements and sentences 413
14.5.1 Conditional phrase 415
14.5.2 Scope of statements 415
14.5.2.1 Explicit scope termination 415
14.5.2.2 Implicit scope termination 415
14.6 Execution 416
14.6.1 Run unit organization 416
14.6.2 State of a function, method, object, or program 416
14.6.2.1 State of a function, method, or program 416
14.6.2.1.1 Active state 417
14.6.2.1.2 Initial and last-used states of data 417
14.6.2.2 Initial state of object data 418
14.6.3 Explicit and implicit transfers of control 418
14.6.4 Item identification 419
14.6.5 Results of runtime element execution 419
14.6.6 Locale identification 420
14.6.7 Sending and receiving operands 420
14.6.8 Alignment of data within data items 421

14.6.9 Overlapping operands . . . 421
14.6.10 Normal run unit termination . . . 422
14.6.11 Abnormal run unit termination . . . 422
14.6.12 Exception condition handling . . . 422
14.6.12.1 Exception conditions . . . 422
14.6.12.1.1 Normal completion of a declarative procedure . . . 424
14.6.12.1.2 Fatal exception conditions . . . 424
14.6.12.1.3 Non fatal exception conditions . . . 424
14.6.12.1.4 Exception objects . . . 425
14.6.12.1.5 Exception-names and exception conditions . . . 426
14.6.12.2 Incompatible data . . . 430
14.7 Common phrases and features for statements . . . 432
14.7.1 At end condition . . . 432
14.7.2 Invalid key condition . . . 432
14.7.3 ROUNDED phrase . . . 432
14.7.4 SIZE ERROR phrase and size error condition . . . 433
14.7.5 CORRESPONDING phrase . . . 435
14.7.6 Arithmetic statements . . . 435
14.7.7 THROUGH phrase . . . 436
14.7.8 RETRY phrase . . . 437
14.8 Conformance for parameters and returning items . . . 438
14.8.1 Parameters . . . 438
14.8.1.1 Group items . . . 438
14.8.1.2 Elementary items . . . 439
14.8.1.2.1 Elementary items passed by reference . . . 439
14.8.1.2.2 Elementary items passed by content or by value . . . 440
14.8.2 Returning items . . . 441
14.8.2.1 Group items . . . 441
14.8.2.2 Elementary items . . . 441
14.9 Statements . . . 443
14.9.1 ACCEPT statement . . . 443
14.9.2 ADD statement . . . 449
14.9.3 ALLOCATE statement . . . 452
14.9.4 CALL statement . . . 454
14.9.5 CANCEL statement . . . 460
14.9.6 CLOSE statement . . . 462
14.9.7 COMPUTE statement . . . 465
14.9.8 CONTINUE statement . . . 467
14.9.9 DELETE statement . . . 468
14.9.10 DISPLAY statement . . . 470
14.9.11 DIVIDE statement . . . 473
14.9.12 EVALUATE statement . . . 476
14.9.13 EXIT statement . . . 481
14.9.14 FREE statement . . . 485
14.9.15 GENERATE statement . . . 486
14.9.16 GO TO statement . . . 488

14.9.17 GOBACK statement 489
14.9.18 IF statement 491
14.9.19 INITIALIZE statement 493
14.9.20 INITIATE statement 497
14.9.21 INSPECT statement 498
14.9.22 INVOKE statement 504
14.9.23 MERGE statement 508
14.9.24 MOVE statement 513
14.9.25 MULTIPLY statement 521
14.9.26 OPEN statement 523
14.9.27 PERFORM statement 528
14.9.28 RAISE statement 534
14.9.29 READ statement 535
14.9.30 RELEASE statement 542
14.9.31 RESUME statement 544
14.9.32 RETURN statement 545
14.9.33 REWRITE statement 547
14.9.34 SEARCH statement 552
14.9.35 SET statement 557
14.9.36 SORT statement 568
14.9.37 START statement 575
14.9.38 STOP statement 579
14.9.39 STRING statement 580
14.9.40 SUBTRACT statement 583
14.9.41 SUPPRESS statement 586
14.9.42 TERMINATE statement 587
14.9.43 UNLOCK statement 588
14.9.44 UNSTRING statement 589
14.9.45 USE statement 593
14.9.46 VALIDATE statement 598
14.9.47 WRITE statement 602

15 Intrinsic functions 610
15.1 General 610
15.2 Types of functions 610
15.3 Arguments 610
15.3.1 Format arguments to date and time functions 612
15.3.1.1 Date formats 612
15.3.1.1.1 Calendar date formats 612
15.3.1.1.2 Permissible values for data associated with calendar date formats 612
15.3.1.1.3 Ordinal date formats 613
15.3.1.1.4 Permissible values for data associated with ordinal date formats 613
15.3.1.1.5 Week date formats 613
15.3.1.1.6 Permissible values for data associated with week date formats 613
15.3.1.2 Time formats 613
15.3.1.2.1 Common time formats 614

15.3.1.2.2 Local time formats . . . 615
15.3.1.2.3 UTC time formats . . . 615
15.3.1.2.4 Offset time formats . . . 615
15.3.1.2.5 Permissible values for data associated with offset time formats . . . 616
15.3.1.3 Combined date and time formats . . . 616
15.3.1.4 Examples of date and time formats . . . 616
15.4 Returned values . . . 618
15.4.1 Numeric and integer functions . . . 618
15.5 Date and time conversion functions . . . 619
15.5.1 Integer date form . . . 619
15.5.2 Standard date form . . . 619
15.5.3 Julian date form . . . 619
15.5.4 Standard numeric time form . . . 619
15.6 Summary of functions . . . 619
15.7 ABS function . . . 626
15.8 ACOS function . . . 627
15.9 ANNUITY function . . . 628
15.10 ASIN function . . . 629
15.11 ATAN function . . . 630
15.12 BOOLEAN-OF-INTEGER function . . . 631
15.13 BYTE-LENGTH function . . . 632
15.14 CHAR function . . . 633
15.15 CHAR-NATIONAL function . . . 634
15.16 COMBINED-DATETIME function . . . 635
15.17 COS function . . . 636
15.18 CURRENT-DATE function . . . 637
15.19 DATE-OF-INTEGER function . . . 638
15.20 DATE-TO-YYYYMMDD function . . . 639
15.21 DAY-OF-INTEGER function . . . 640
15.22 DAY-TO-YYYYDDD function . . . 641
15.23 DISPLAY-OF function . . . 642
15.24 E function . . . 643
15.25 EXCEPTION-FILE function . . . 644
15.26 EXCEPTION-FILE-N function . . . 645
15.27 EXCEPTION-LOCATION function . . . 646
15.28 EXCEPTION-LOCATION-N function . . . 647
15.29 EXCEPTION-STATEMENT function . . . 648
15.30 EXCEPTION-STATUS function . . . 649
15.31 EXP function . . . 650
15.32 EXP10 function . . . 651
15.33 FACTORIAL function . . . 652
15.34 FORMATTED-CURRENT-DATE function . . . 653
15.35 FORMATTED-DATE function . . . 654
15.36 FORMATTED-DATETIME function . . . 655
15.37 FORMATTED-TIME function . . . 656
15.38 FRACTION-PART function . . . 657

15.39 HIGHEST-ALGEBRAIC function 658
15.40 INTEGER function 659
15.41 INTEGER-OF-BOOLEAN function 660
15.42 INTEGER-OF-DATE function 661
15.43 INTEGER-OF-DAY function 662
15.44 INTEGER-OF-FORMATTED-DATE function 663
15.45 INTEGER-PART function 664
15.46 LENGTH function 665
15.47 LOCALE-COMPARE function 667
15.48 LOCALE-DATE function 668
15.49 LOCALE-TIME function 669
15.50 LOCALE-TIME-FROM-SECONDS function 670
15.51 LOG function 671
15.52 LOG10 function 672
15.53 LOWER-CASE function 673
15.54 LOWEST-ALGEBRAIC function 674
15.55 MAX function 675
15.56 MEAN function 676
15.57 MEDIAN function 677
15.58 MIDRANGE function 678
15.59 MIN function 679
15.60 MOD function 680
15.61 NATIONAL-OF function 681
15.62 NUMVAL function 682
15.63 NUMVAL-C function 683
15.64 NUMVAL-F function 685
15.65 ORD function 686
15.66 ORD-MAX function 687
15.67 ORD-MIN function 688
15.68 PI function 689
15.69 PRESENT-VALUE function 690
15.70 RANDOM function 691
15.71 RANGE function 692
15.72 REM function 693
15.73 REVERSE function 694
15.74 SECONDS-FROM-FORMATTED-TIME function 695
15.75 SECONDS-PAST-MIDNIGHT function 696
15.76 SIGN function 697
15.77 SIN function 698
15.78 SQRT function 699
15.79 STANDARD-COMPARE function 700
15.80 STANDARD-DEVIATION function 701
15.81 SUM function 702
15.82 TAN function 703
15.83 TEST-DATE-YYYYMMDD function 704
15.84 TEST-DAY-YYYYDDD function 705

15.85 TEST-FORMATTED-DATETIME function . . . 706
15.86 TEST-NUMVAL function . . . 707
15.87 TEST-NUMVAL-C function . . . 708
15.88 TEST-NUMVAL-F function . . . 709
15.89 TRIM function . . . 710
15.90 UPPER-CASE function . . . 711
15.91 VARIANCE function . . . 712
15.92 WHEN-COMPILED function . . . 713
15.93 YEAR-TO-YYYY function . . . 714

16 Standard classes . . . 715
16.1 General . . . 715
16.2 BASE class . . . 715
16.2.1 New method . . . 715
16.2.2 FactoryObject method . . . 715

A Language element lists . . . 717
A.1 Implementor-defined language element list . . . 717
A.2 Undefined language element list . . . 731
A.3 Processor-dependent language element list . . . 736
A.4 Optional language element list . . . 738

B Characters permitted in user-defined words . . . 743
B.1 General . . . 743
B.2 Notation . . . 743
B.3 Repertoire of characters permitted in user-defined words . . . 743

C Mapping of uppercase letters to lowercase letters . . . 747
C.1 Notations . . . 747
C.2 General case mappings . . . 747
C.3 Additional case mappings . . . 750

D Concepts . . . 751
D.1 General . . . 751
D.2 Files . . . 751
D.3 Tables and any-length elementary items . . . 760
D.4 Shared memory area . . . 768
D.5 Sharing of storage among data items . . . 768
D.6 Compilation group and run unit organization and communication . . . 770
D.7 Intrinsic function facility . . . 784
D.8 Types . . . 785
D.9 Addresses and pointers . . . 787
D.10 Boolean support and bit manipulation . . . 789
D.11 Character sets . . . 792
D.12 Collating sequences . . . 795
D.13 Culturally-specific, culturally-adaptable, and multilingual applications . . . 798

D.14 External switches 803
D.15 Common exception processing 803
D.16 Rounding 805
D.17 Forms of arithmetic 807
D.18 Object oriented concepts 816
D.19 Report writer 835
D.20 Structured constant 842
D.21 Validate facility 843
D.22 Conditional expressions 846
D.23 Examples of the execution of the INSPECT statement 850
D.24 Examples of the execution of the PERFORM statement 853
D.25 Example of free-form reference format 857
D.26 Conditional compilation 858
D.27 CALL-CONVENTION directive 859
D.28 ENTRY-CONVENTION clause 859
D.29 Date and time handling 859

E Substantive changes list 865
E.1 General 865
E.2 Substantive changes potentially affecting existing programs 865
E.3 Substantive changes not affecting existing programs 870

F Archaic and obsolete language element lists 873
F.1 Archaic language elements 873
F.2 Obsolete language elements 874

G Known errors in the standard 875
G.1 Rationale 875
G.2 List of errors 875

H Bibliography 876

Index 877

Tables

1 Verbal forms 22
2 COBOL character repertoire 63
3 Class and category relationships for elementary data items 120
4 Combinations of symbols in arithmetic expressions 131
5 Combination of symbols in boolean expressions 140
6 Combinations of conditions, logical operators, and parentheses 154
7 Relationship of alphabet-name to coded character set and collating sequence 225
8 Category and type of editing 349
9 Results of fixed insertion editing 350
10 Results of floating insertion editing 351
11 Format 1 picture symbol order of precedence 354
12 Format 2 picture symbol order of precedence 355
13 Procedural statements 413
14 Exception-names and exception conditions 426
15 Relationship of categories of physical files and the format of the CLOSE statement 462
16 Combination of operands in the EVALUATE statement 478
17 Validity of types of MOVE statements 514
18 Category of figurative constants used in the MOVE statement 519
19 Opening available and unavailable files (file not currently open) 524
20 Opening available shared files that are currently open by another file connector 524
21 Permissible I-O statements by access mode and open mode 525
22 Table of date and time formats 616
23 Table of functions 620
C.1 Summary of record lock acquisition and release 759
C.2 Examples of boolean operations 790
C.3 ROUNDED MODE examples 806

Figures

1 Fixed-form reference format 25
C.1 Format 1 SEARCH statement having two WHEN phrases 764
C.2 Compilation group sample structure example 772
C.3 Compilation group and run unit structures 773
C.4 Manager class 821
C.5 Banking hierarchy 822
C.6 Example of page layout 836
C.7 Evaluation of the condition-1 AND condition-2 AND ... condition-n 847
C.8 Evaluation of the condition-1 OR condition-2 OR ... condition-n 847
C.9 Evaluation of condition-1 OR condition-2 AND condition-3 848
C.10 Evaluation of (condition-1 OR NOT condition-2) AND condition-3 AND condition-4 849
C.11 The VARYING phrase of a PERFORM statement with the TEST BEFORE phrase having one condition . . . 853
C.12 The VARYING phrase of a PERFORM statement with the TEST BEFORE phrase having two conditions . . 854
C.13 The VARYING phrase of a PERFORM statement with the TEST AFTER phrase having one condition 855
C.14 The VARYING phrase of a PERFORM statement with the TEST AFTER phrase having two conditions 856

Foreword

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

International standards are drafted in accordance with the rules given in the ISO/IEC Directives, part 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.

A complete list of technical changes is given in Annex E.

Annexes A, B, and C form a normative part of this final committee draft International Standard. Annexes D through H are for information only.

Introduction

COBOL began as a business programming language, but its present use has spread well beyond that to a general purpose programming language. Significant enhancements in this final committee draft International Standard include:

- Dynamic-capacity tables
- Function pointers
- Any-length elementary items
- Increased size limit on nonnumeric literals
- Enhanced locale support in functions
- Support for industry standard floating-point formats
- Support for industry-standard arithmetic rules
- Support for multiple rounding options
- Structured constants
- Support for industry standard date and time formats
- Parametric polymorphism, also known as method overloading

Annex D, Concepts, includes an explanation of major features as well as the more complicated prior features and is the suggested starting point for the reading of this document.

The previous COBOL standard was published in 2002. Implementors have provided language extensions in response to the demands of their users. Several changes and extensions have, therefore, been made in this final committee draft International Standard to prevent further divergence, and to ensure consistency among, and coherence within, various implementations.

Development of the COBOL language began before the invention of formal techniques for specification of programming languages. Hence, the COBOL standard uses its own description techniques, which are described in 5, Description techniques. These techniques involve general formats, which describe the syntax, and natural language.

This final committee draft International Standard is a result of the standardization efforts of working group ISO/IEC JTC1/SC22/WG4 and INCITS Task Group PL22. During the development of this final committee draft International Standard, great care was taken to minimize changes that would impact existing programs. Most substantive changes that potentially impact existing programs were introduced to resolve ambiguities in the previous COBOL standard. Details of the substantive changes are given in Annex E, Substantive changes list.

Information technology — Programming languages, their environments and system software interfaces — Programming language COBOL

1 Scope

This final committee draft International Standard specifies the syntax and semantics of COBOL. Its purpose is to promote a high degree of machine independence to permit the use of COBOL on a variety of data processing systems.

This final committee draft International Standard specifies:

— The form of a compilation group written in COBOL.

— The effect of compiling a compilation group.

— The effect of executing run units.

— The elements of the language for which a conforming implementation is required to supply a definition.

— The elements of the language for which meaning is explicitly undefined.

— The elements of the language that are dependent on the capabilities of the processor.

This final committee draft International Standard does not specify:

— The means whereby a compilation group written in COBOL is compiled into code executable by a processor.

— The time at which method, function, or program runtime modules are linked or bound to an activating statement, except that runtime binding occurs of necessity when the identification of the appropriate program or method is not known at compile time.

— The time at which parameterized classes and interfaces are expanded.

— The mechanism by which locales are defined and made available on a processor.

— The form or content of error, flagging, or warning messages.

— The form and content of listings produced during compilation, if any.

— The form of documentation produced by an implementor of products conforming to this final committee draft International Standard.

— The sharing of resources other than files among run units.

2 Normative references

The following referenced documents 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.

IIEEE Std 754 2008, *IEEE Standard for Floating-Point Arithmetic*

NOTE While this final committee draft International Standard makes reference to IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, both for the formats of certain user-defined and intermediate data items and for rules associated with arithmetic manipulation of data in these formats, this final committee draft International Standard does not claim conformance to all specifications and rules contained in that floating-point standard.

ISO/IEC 646, *Information technology — ISO 7-bit coded character set for information interchange.*

ISO 1001:1986, *Information processing — File structure and labelling of magnetic tapes for information interchange.*

ISO 8601:2004, *Data elements and interchange formats — Information interchange — Representation of dates and times.*

ISO/IEC 9945:2009, *Information technology — Portable Operating System Interface (POSIX®) Base Specifications, issue 7.*

ISO/IEC 10646, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) (including AMD1 through AMD6).*

ISO/IEC 14651:2007, *Information technology — International string ordering and comparison — Method for comparing character strings and description of the common template tailorable ordering (including AMD1:2009).*

3 Conformance to this final committee draft International Standard

3.1 General

Clause 3 specifies requirements that an implementation shall fulfill in order to conform to this final committee draft International Standard and defines the conditions under which a compilation group or run unit conforms in its use of standard features.

3.2 A conforming implementation

To conform to this final committee draft International Standard, an implementation of standard COBOL shall provide the required normative elements specified in 6, Reference format through 16, Standard classes: optionally provide the normative elements identified in A.4, Optional language element list, and meet the criteria of 3.2.1 through 3.2.16.

3.2.1 Acceptance of standard language elements

An implementation shall accept the syntax and provide the functionality for all standard language elements required by this final committee draft International Standard and the optional or processor-dependent language elements for which support is claimed.

An implementation shall provide a warning mechanism that optionally can be invoked by the user at compile time to indicate violations of the general formats and the explicit syntax rules of standard COBOL. This warning mechanism shall provide a suboption for selection or suppression of checking for violations of the set of conformance rules specified in 14.8, Conformance for parameters and returning items, and in 9.3.8.1.2, Conformance between interfaces.

There are rules in standard COBOL that are not identified as general formats or syntax rules, but nevertheless specify elements that are syntactically distinguishable. This warning mechanism shall indicate violations of such rules. For elements not specified in general formats or in explicit syntax rules, it is left to the implementor's judgement to determine what is syntactically distinguishable.

There are general rules in standard COBOL that could have been classified as syntax rules. These rules are classified as general rules for the purpose of avoiding syntax checking, and do not reflect errors in standard COBOL. An implementation may, but is not required to, flag violations of such rules.

3.2.2 Interaction with non-COBOL runtime modules

Facilities are provided in this specification that enable transfer of control and sharing of external items between COBOL runtime modules and non-COBOL runtime modules. No requirement is placed on an implementation to support this interaction. When supported, an implementor shall document the languages and the implementations supported.

3.2.3 Interaction between COBOL implementations

Facilities are provided in this specification that enhance the capability of transferring control and sharing external items between COBOL runtime elements translated on COBOL implementations produced by different implementors. No requirement is placed on an implementation to support this interaction. When supported, an implementor shall document the implementations supported.

3.2.4 Implementor-defined language elements

The provisions of this clause apply to required normative elements of this final committee draft International Standard and to optional language elements for which an implementor claims support.

Language elements that depend on implementor definition to complete the specification of the syntax rules or general rules are listed in A.1, Implementor-defined language element list. To meet the requirements of standard COBOL, the implementor shall specify, at a minimum, the implementor-defined language elements that are

identified as required. Each implementor-defined language element specified by the implementor shall be documented if the implementor-defined language element is identified as requiring user documentation.

An implementor shall not require the inclusion of nonstandard language elements in a compilation group as part of the definition of an implementor-defined language element.

3.2.5 Processor-dependent language elements

Processor refers to the entire computing system that is used to translate compilation groups and execute run units, consisting of both hardware and relevant software. Language elements that depend on specific devices or on a specific processor capability, functionality, or architecture are listed in A.3, Processor-dependent language element list. To meet the requirements of standard COBOL, the implementor shall document the processor-dependent language elements for which the implementation claims support. Language elements that pertain to specific processor-dependent elements for which support is not claimed need not be implemented. The decision of whether to claim support for a processor-dependent language element is within an implementor's discretion. Factors that may be considered include, but are not limited to, hardware capability, software capability, and market positioning of the processor.

When standard-conforming support is claimed for a specific processor-dependent language element, all associated syntax and functionality required for that language element shall be implemented; when a subset of the syntax or functionality is implemented, that subset shall be identified as a standard extension in the implementor's user documentation. The absence of processor-dependent elements from an implementation shall be specified in the implementor's user documentation.

An implementation shall provide a warning mechanism at compile time to indicate use of syntactically-detectable processor-dependent language elements not supported by that implementation. Although this warning mechanism is required to identify processor-dependent elements that are not supported, it is not required to diagnose syntax errors within this unsupported syntax. The implementor is not required to produce executable code when unsupported processor-dependent language elements are used.

3.2.6 Optional language elements

Language elements that an implementor may, but need not, implement are listed in A.4, Optional language element list. An implementor shall identify in user documentation the optional language elements for which that implementor claims support. If an implementor provides support for parts of an optional feature, user documentation shall identify the elements that are supported and those that are not supported. The provisions of 3.2.4, Implementor-defined language elements, apply for each optional language element for which support is claimed.

3.2.7 Reserved words

An implementation shall recognize as reserved words all the COBOL reserved words specified in 8.9, Reserved words; shall recognize in context all the context-sensitive words specified in 8.10, Context-sensitive words; and shall recognize in compiler directives all the compiler-directive words specified in 8.12, Compiler-directive words.

3.2.8 Standard extensions

An implementor may claim support for all or a subset of the syntax and associated functionality of optional or processor-dependent elements. When an implementor claims support for a subset of the syntax, that syntax is a 'standard extension', provided that the associated functionality is that specified in this final committee draft International Standard. If different functionality is provided, that syntax is a nonstandard extension.

3.2.9 Nonstandard extensions

Nonstandard extensions are language elements or functionality in an implementation that consist of any of the following:

1) documented language elements not defined in this final committee draft International Standard;

2) language elements defined in this final committee draft International Standard for which different functionality is implemented, where that language element is not required for conformance to this final committee draft International Standard, and standard support for that element is not claimed by the implementor;

3) language elements defined in this final committee draft International Standard for which different functionality is implemented, where that language element is required for conformance to this final committee draft International Standard, provided that standard-conforming behavior is also implemented and that an implementor-defined mechanism exists for selection of the nonstandard behavior.

An implementation that introduces additional reserved words as nonstandard extensions conforms to this final committee draft International Standard, even though the additional reserved words may prevent translation of some conforming compilation groups.

Documentation associated with an implementation shall identify nonstandard extensions for which support is claimed and shall specify any reserved words added for nonstandard extensions.

An implementation shall provide a warning mechanism that optionally can be invoked by the user at compile time to indicate use of a nonstandard extension in a compilation group. This warning mechanism is required to flag only extensions that are syntactically distinguishable.

3.2.10 Substitute or additional language elements

An implementation shall not require the inclusion of substitute or additional language elements in the compilation group in order to accomplish functionality specified for a standard language element.

3.2.11 Archaic language elements

Archaic language elements are those identified in F.1, Archaic language elements. Archaic language elements should not be used in new compilation groups because better programming practices exist. There is no schedule for deleting archaic elements from standard COBOL; however, this may be reevaluated for any future editions of standard COBOL.

An implementation is required to support archaic language elements of the facilities for which support is claimed. Documentation associated with an implementation shall identify archaic language elements in the implementation.

An implementation shall provide a warning mechanism that optionally can be invoked by the user at compile time to indicate use of an archaic language element in a compilation group.

3.2.12 Obsolete language elements

Obsolete language elements are identified in F.2, Obsolete language elements. Obsolete language elements will be removed from the next edition of standard COBOL.

An implementation is required to support obsolete language elements of the facilities for which support is claimed. Documentation associated with an implementation shall identify all obsolete language elements in the implementation.

An implementation shall provide a warning mechanism that optionally can be invoked by the user at compile time to indicate use of an obsolete element in a compilation group.

3.2.13 Externally-provided functionality

An implementation may require specifications outside the compilation group to interface with the operating environment to support functionality specified in a compilation group.

An implementation may require the presence in the operating environment of runtime modules or products in addition to the COBOL implementation to support syntax or functionality specified in a compilation group.

NOTE This permits an implementation to require components outside the COBOL implementation, such as pre-compilers, file systems, and sort products.

3.2.14 Limits

In general, standard COBOL specifies no upper limit on such things as the number of statements in a compilation group or the number of operands permitted in certain statements. A conforming implementation may place such limits. It is recognized that these limits will vary from one implementation of standard COBOL to another and may prevent the successful translation by a conforming implementation of some compilation groups that meet the requirements of standard COBOL.

3.2.15 User documentation

An implementation shall satisfy the user documentation requirements specified in 3.2.2, 3.2.3, 3.2.4, 3.2.5, 3.2.9, 3.2.11, and 3.2.12 by specification in at least one form of documentation. This may include, but is not limited to, hardcopy manuals, on-line documentation, and user help screens.

Documentation requirements may be met by reference to other documents, including those of the operating environment and other COBOL implementations.

3.2.16 Character substitution

The definition of the COBOL character repertoire in 8.1.2, COBOL character repertoire, presents the complete COBOL character repertoire for standard COBOL. When an implementation does not provide a graphic representation for all the basic characters of the COBOL character repertoire, substitute graphics may be specified by the implementor to replace the characters not represented.

3.3 A conforming compilation group

A conforming compilation group is one that does not violate the explicitly stated syntactic provisions and specifications of standard COBOL. In order for a compilation group to conform to standard COBOL, it shall not include any language elements not specified in this final committee draft International Standard. A compilation group that uses elements that are optional, processor-dependent, or implementor-defined in this final committee draft International Standard is a conforming compilation group, even on implementations where it does not compile successfully due to the use of those elements.

The compilation units contained in a conforming compilation group are conforming compilation units

3.4 A conforming run unit

A conforming run unit is one that:

1) is composed of one or more runtime modules, each resulting from a successful compilation of a conforming compilation unit, and

2) complies with the explicitly stated provisions and specifications of standard COBOL with respect to the runtime behavior of that run unit.

NOTE The inclusion of non-COBOL components in the run unit does not affect the conformance of the run unit.

The processing of a conforming run unit is predictable only to the extent defined in standard COBOL. The results of violating the formats or rules of standard COBOL are undefined unless otherwise specified in this final committee draft International Standard.

Situations in which the results of executing a statement are explicitly undefined or unpredictable are identified in A.2, Undefined language element list. A COBOL run unit that allows these situations to happen is a conforming run unit, although the resultant execution is not defined by standard COBOL.

3.5 Relationship of a conforming compilation group to a conforming implementation

The translation of a conforming compilation group by a conforming implementation is defined only to the extent specified in standard COBOL. It is possible that a conforming compilation group will not be translated successfully. Translation may be unsuccessful due to factors other than lack of conformance of a compilation group.

NOTE These factors can include the use of optional, processor-dependent, or implementor-defined language elements and the limits of an implementation.

3.6 Relationship of a conforming run unit to a conforming implementation

The execution of a run unit composed of runtime modules resulting from translation of conforming compilation units is defined only to the extent specified in standard COBOL. It is possible that a conforming run unit will not execute successfully. Execution may be unsuccessful due to factors other than lack of conformance of a run unit.

NOTE These factors can include the logical incorrectness of the compilation units, errors in the data upon which the run unit operates, and the limits of an implementation.

4 Terms and definitions

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

4.1 **absolute item:** An item in a report that has a fixed position on a page.

4.2 **activated runtime element:** A function, method, or program placed into the active state.

4.3 **activating statement:** A statement that causes the execution of a function, method, or program.

4.4 **activating runtime element:** The function, method, or program that executed a given activating statement.

4.5 **active state:** The state of a function, method, or program that has been activated but has not yet returned control to the activating runtime element.

4.6 **alphabetic character (in the COBOL character repertoire):** A basic letter or a space character.

4.7 **alphanumeric character:** Any coded character in an alphanumeric coded character set, whether or not there is an assigned graphic symbol for that coded character.

4.8 **alphanumeric character position:** The amount of physical storage required to store, or presentation space required to print or display, a single character of an alphanumeric character set.

4.9 **alphanumeric character set; alphanumeric coded character set:** See alphanumeric coded character set.

4.10 **alphanumeric coded character set; alphanumeric character set:** A coded character set that the implementor has designated for representation of data items of usage display and alphanumeric literals.

4.11 **alphanumeric group item:** Any group item except for:

— a bit group item
— a national group item
— a strongly-typed group item
— a variable-length group item.

4.12 **argument:** An operand specified in an activating statement that specifies the data to be passed.

4.13 **assumed decimal point:** A decimal point position that does not involve the existence of an actual character in a data item. An assumed decimal point has logical meaning with no physical representation.

4.14 **based data item:** A data item established by association of a based entry with an actual data item or allocated storage.

4.15 **based entry:** A data description entry that serves as a template that is dynamically associated with data items or allocated storage.

4.16 **basic letter:** One of the uppercase letters 'A' through 'Z' and the lowercase letters 'a' through 'z' in the COBOL character repertoire.

4.17 **bit:** The smallest unit in a computer's storage structure capable of representing two distinct alternatives.

4.18 **bit data item:** An elementary data item of category boolean and usage bit or a bit group item.

4.19 **block; physical record:** A physical unit of data that is normally composed of one or more logical records.

4.20 **boolean character:** A unit of information that consists of the value zero or one. Each boolean character may be represented in storage as a bit, an alphanumeric character, or a national character.

4.21 **boolean data item:** A data item capable of containing a boolean value.

4.22 **boolean expression:** One or more boolean operands separated by boolean operators.

4.23 **boolean position:** The amount of physical storage required to store, or presentation space required to print or display, a single boolean character.

4.24 **boolean value:** A value consisting of a sequence of one or more boolean characters.

4.25 **byte:** A sequence of bits representing the smallest addressable character unit in the memory of a given computer.

4.26 **case-sensitive:** Of, or relating to, the treatment of uppercase letters and their corresponding lowercase letters as having meanings or interpretations that are equivalent.

4.27 **case-insensitive:** Of, or relating to, the treatment of uppercase letters and their corresponding lowercase letters as having meanings or interpretations that are distinct from each other.

4.28 **character (in a coded character set):** A code value that constitutes the coded representation of a letter, digit, symbol, control function, or other member of a set of elements used for the organization, control, or representation of data.

NOTE COBOL processes each code value in a coded character set as though it were a character.

4.29 **character (in a screen item):** A graphic character.

4.30 **character (in COBOL character repertoire); COBOL character:** A letter, digit, or special character, independent of its coded representation, that is used in formation of the words and separators of COBOL.

4.31 **character (in computer storage):** A single code value of a coded character set.

4.32 **character boundary:** The leftmost bit of an addressing boundary in the storage of the computer.

4.33 **character position:** The amount of physical storage required to store, or presentation space required to print or display, a single character -- either an alphanumeric character or a national character. One element of a coded character set occupies one character position.

NOTE As an example, each element of a combining sequence in the UCS occupies one character position. For a UTF-16 surrogate value, the left surrogate occupies one character position and the right surrogate occupies another character position.

4.34 **character-string:** A sequence of contiguous characters that form a COBOL word, a literal, or a picture character-string.

4.35 **class (in object orientation)**: The entity that defines common behavior and implementation for zero, one, or more objects.

4.36 **class (of a data item)**: A designation for a set of data items having common attributes or a common range of values, defined by the PICTURE clause, the USAGE clause, or the PICTURE and USAGE clauses in a data description entry; by the definition of a predefined identifier; or by the definition of an intrinsic function.

4.37 **class (of data values)**: A designation for a set of data values that are permissible in the content of a data item.

4.38 **class definition (in object orientation)**: A compilation unit that defines a class of objects.

4.39 **clause:** An ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

4.40 **COBOL character; character (in COBOL character repertoire):** See character (in COBOL character repertoire).

4.41 **COBOL character repertoire:** The repertoire of characters used in writing the syntax of a COBOL compilation group, except for comments and the content of non-hexadecimal alphanumeric and national literals.

4.42 **coded character set:** A set of unambiguous rules that establishes a character set and the relationship between the characters of the set and their coded representation. [ISO/IEC 10646]

4.43 **combining character:** A UCS member that is intended for combination with the preceding non-combining graphic character or with a sequence of combining characters preceded by a non-combining character.

4.44 **common program:** A program that, despite being directly contained within another program, may be called from any program directly or indirectly contained in that other program.

4.45 **compilation group:** A sequence of one or more compilation units submitted together for compilation.

4.46 **compilation unit:** A source unit that is not nested within another source unit.

4.47 **composite sequence:** A sequence of graphic characters consisting of a non-combining character followed by one or more combining characters. [ISO/IEC 10646].

4.48 **conditional statement:** A statement for which the truth value of a specified condition is evaluated and used to determine subsequent flow of control.

4.49 **conformance (for object orientation):** A unidirectional relation that allows an object to be used according to an interface other than the interface of its own class.

4.50 **conformance (for parameters):** The requirements for the relationship between arguments and formal parameters and between returning items in activating and activated runtime elements.

4.51 **control function:** An action that affects the recording, processing, transmission, or interpretation of data, and that has a coded representation consisting of one or more bytes.

NOTE This definition is the same as that in ISO/IEC 10646 except that 'octets' is replaced by 'bytes' because the term 'octet' is not used in the COBOL specification.

4.52 **cultural element:** An element of data for computer use that may vary dependent on language, geographical territory, or other cultural circumstances.

4.53 **currency sign:** The COBOL character '$', used as the default currency symbol in a picture character-string and as the default currency string that appears in the edited format of data items.

NOTE Features exist for selection of other currency strings and currency symbols.

4.54 **currency string:** The set of characters to be placed into numeric-edited data items as a result of editing operations when the item includes a currency symbol in its picture character-string.

4.55 **currency symbol:** The character used in a picture character-string to represent the presence of a currency string.

4.56 **current record:** The record that is available in the record area associated with a file.

4.57 **current volume pointer:** A conceptual entity that points to the current volume of a sequential file.

4.58 **data item:** A unit of data defined by a data description entry or resulting from the evaluation of an identifier.

4.59 **decimal point; decimal separator:** The character used to represent the radix point. The default is the character period.

4.60 **decimal separator; decimal point:** See decimal point.

4.61 **declarative statement:** A USE statement, which defines the conditions under which the procedures that follow the statement will be executed.

4.62 **de-editing:** The logical removal of all editing characters from a numeric-edited data item in order to determine that item's unedited numeric value.

4.63 **delimited scope statement:** Any statement that is terminated by its explicit scope terminator.

4.64 **digit position:** The amount of physical storage required to store, or presentation space required to print or display, a single digit.

4.65 **dynamic access:** An access mode in which specific logical records may be obtained from or placed into a mass storage file in a nonsequential manner and obtained from a file in a sequential manner.

4.66 **dynamic storage:** Storage that is allocated and released on request during runtime.

4.67 **end marker**: A marker for the end of a source unit.

4.68 **entry:** A descriptive set of consecutive clauses terminated by a separator period.

4.69 **entry convention:** The information required to interact successfully with a function, method, or program.

4.70 **exception condition:** A condition detected at runtime that indicates that an error or exception to normal processing has occurred.

4.71 **exception object**: An object that acts as an exception condition.

4.72 **exception status indicator:** A conceptual entity that exists for each exception-name.

4.73 **EXIT FUNCTION statement:** an abbreviation for 'EXIT statement with the FUNCTION phrase'.

4.74 **EXIT METHOD statement:** an abbreviation for 'EXIT statement with the METHOD phrase'.

4.75 **EXIT PARAGRAPH statement:** an abbreviation for 'EXIT statement with the PARAGRAPH phrase'.

4.76 **EXIT PERFORM statement:** an abbreviation for 'EXIT statement with the PERFORM phrase'.

4.77 **EXIT PROGRAM statement:** an abbreviation for 'EXIT statement with the PROGRAM phrase'.

4.78 **EXIT SECTION statement:** an abbreviation for 'EXIT statement with the SECTION phrase'.

4.79 **explicit scope terminator:** A statement-dependent word that by its presence terminates the scope of that statement.

4.80 **extend mode:** A mode of file processing in which records may be added at the end of a sequential file, but no records may be deleted, read, or updated.

4.81 **extended letter:** A letter, other than the basic letters, in the set of characters defined for the COBOL character repertoire.

4.82 **external data:** Data that belongs to the run unit and may be accessed by any runtime element in which it is described.

4.83 **external media format:** A form of data suitable for presentation or printing, including any control functions necessary for representation as readable text.

4.84 **external switch:** A hardware or software device, defined and named by the implementor, that is used to indicate that one of two alternate states exists.

4.85 **factory object:** The single object associated with a class, defined by the factory definition of that class, typically used to create the instance objects of the class.

4.86 **file:** A logical entity that represents a collection of logical records. There is one logical file associated with one file connector and there may be several logical files associated with one physical file.

4.87 **file connector:** A storage area that contains information about a file and is used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

4.88 **file organization:** The permanent logical file structure established at the time that a file is created.

4.89 **file position indicator:** A conceptual entity that either is used to facilitate exact specification of the next record to be accessed, or indicates why such a reference cannot be established.

4.90 **file sharing:** A cooperative environment that controls concurrent access to the same physical file.

4.91 **fixed file attribute:** An attribute of a physical file that is established when the physical file is created and is never changed during the existence of the physical file.

4.92 **formal parameter:** A data-name specified in the USING phrase of the procedure division header that gives the name used in the function, method, or program for a parameter.

4.93 **function:** An intrinsic or user-defined procedural entity that returns a value based upon the arguments.

4.94 **function prototype definition:** A definition that specifies the rules governing the arguments needed for the evaluation of a particular function, the data item resulting from the evaluation of the function, and all other requirements needed for the evaluation of that function.

4.95 **graphic character:** A character, other than a control function, that has a visual representation normally handwritten, printed, or displayed. [ISO/IEC 10646].

4.96 **graphic symbol:** The visual representation of a graphic character or of a composite sequence. [ISO/IEC 10646].

4.97 **grouping (in locale editing):** The separation of digits into groups in number and currency formatting.

4.98 **grouping separator:** The character used to separate digits in numbers for ease of reading. The default is the character comma.

4.99 **high-order end:** The leftmost position of a string of characters or a string of bits.

4.100 **i-o mode:** A mode of file processing in which records can be read, updated, added, and deleted.

4.101 **i-o status:** A conceptual entity that exists for a file, that contains a value indicating the result of the execution of an input-output operation for that file.

4.102 **IEEE binary floating-point usage:** a collective reference to the usages float-binary-7, float-binary-16, and float-binary-34, as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings, as binary32, binary64, and binary128 respectively.

4.103 **IEEE decimal floating-point usage:** a collective reference to the usages float-decimal-16 and float-decimal-34, as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Decimal interchange format encodings, as decimal64, and decimal128 respectively.

4.104 **IEEE floating-point usage:** a collective reference to the IEEE binary and IEEE decimal floating-point usages.

4.105 **imperative statement:** A statement that specifies an unconditional action or that is a delimited scope statement.

4.106 **index:** A storage area or a register, the content of which refers to a particular element in a table.

4.107 **indexed organization:** The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

4.108 **inheritance (for classes):** A mechanism for using the interface and implementation of one or more classes as the basis for another class.

4.109 **inheritance (for interfaces):** A mechanism for using the specification of one or more interfaces as the basis for another interface.

4.110 **initial program:** A program that is placed into the initial state every time the program is called.

4.111 **initial state:** The state of a function, method, or program when it is first activated in a run unit.

4.112 **input mode:** A mode of file processing in which records can only be read.

4.113 **instance object:** A single instance of an object defined by a class and created by a factory object.

4.114 **interface (of an object):** The names of all the methods defined for the object, including inherited methods; for each of the methods:

- the ordered list of its formal parameters and the description and passing technique associated with each
- any returned value and its description
- exceptions that may be raised.

4.115 **interface (the language construct)**: A grouping of method prototypes.

4.116 **internal data:** All data described in a source unit except external data and external file connectors.

4.117 **invocation; method invocation**: See method invocation.

4.118 **key of reference:** The key, either prime or alternate, currently being used to access records within an indexed file.

4.119 **letter:** A basic letter or an extended letter.

4.120 **locale:** A facility in the user's information technology environment that specifies language and cultural conventions.

4.121 **lock mode:** The state of a file for which record locking is in effect that indicates whether record locking is manual or automatic.

4.122 **low-order end:** The rightmost position of a string of characters or a string of bits.

4.123 **method**: A procedural entity defined by a method definition within, or inherited by, a class definition as an allowable operation upon objects of that class.

4.124 **method data**: Data declared in a method definition.

4.125 **method invocation; invocation**: The request to execute a named method on a given object.

4.126 **method prototype**: A source element that specifies the information needed for invocation of a method and for conformance checking.

4.127 **national character:** A character in a national character set.

4.128 **national character position:** The amount of physical storage required to store, or presentation space required to print or display, a single national character.

4.129 **national character set; national coded character set:** See national coded character set.

4.130 **national coded character set; national character set:** A coded character set that the implementor has designated for representation of data items described as usage national and for national literals.

4.131 **national data item:** An elementary data item of class national or a national group item.

4.132 **native alphanumeric character set:** The computer's alphanumeric coded character set.

4.133 **native arithmetic:** A mode of arithmetic in which the techniques used in handling arithmetic are specified by the implementor.

4.134 **native character set:** An implementor-defined character set, either alphanumeric or national or both, that is used for internal processing of a COBOL runtime module. The native character set is that referenced by the keyword NATIVE in the SPECIAL-NAMES paragraph.

4.135 **native collating sequence:** An implementor-defined collating sequence, either an alphanumeric collating sequence or a national collating sequence, that is associated with the computer on which a runtime module is executed.

4.136 **native national coded character set:** The computer's national coded character set.

4.137 **next record:** The record that logically follows the current record of a file.

4.138 **null:** The state of a pointer indicating that it contains no address, or the state of an object reference indicating that it contains no reference.

4.139 **numeric character (in the rules of COBOL):** A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

4.140 **object:** A unit consisting of data and the methods that act upon that data.

4.141 **object data:** Data defined:

— in the factory definition, except for the data defined in its methods.
— in the instance definition, except for the data defined in its methods.

4.142 **object property; property**: A name that may be used to qualify an object reference to get a value from or pass a value to an object.

4.143 **object reference**: An explicitly- or implicitly-defined data item that contains a reference to an object.

4.144 **open mode:** The state of a file connector indicating input-output operations that are permitted for the associated file.

4.145 **optional file:** A file declared as being not necessarily present each time the runtime module is executed.

4.146 **outermost program:** A program, together with its contained programs, that is not contained in any other program.

4.147 **output file:** A file that is opened in either the output mode or extend mode.

4.148 **output mode:** A mode of file processing in which a file is created and records can only be added to the file.

4.149 **physical file:** A physical collection of physical records.

4.150 **physical record; block:** See block.

4.151 **previous record:** The record that logically precedes the current record of a file.

4.152 **procedure:** One or more successive paragraphs or sections in the procedure division.

4.153 **procedure branching statement:** A statement that causes the explicit transfer of control to a statement other than the next executable statement in the sequence in which the statements are written.

4.154 **processor:** The computing system, both hardware and software, used for compilation of source code or execution of run units, or both.

4.155 **program prototype definition:** A definition that specifies the rules governing the class of the parameters expected to be received by a particular called program, and any other requirements needed to transfer control to and get control and return information from that called program.

4.156 **property; object property:** See object property.

4.157 **random access:** An access mode in which the value of a key data item identifies the logical record that is obtained from, deleted from, or placed into a relative or indexed file.

4.158 **record key:** A data item within a record used to identify that record within an indexed file.

4.159 **record lock:** A conceptual entity that is used to control concurrent access to a given record within a shared physical file.

4.160 **record locking:** A facility for controlling concurrent access to records in a shared physical file.

4.161 **relative item:** An item in a report whose position is specified relative to the previous item.

4.162 **relative key:** A data item that contains a relative record number.

4.163 **relative organization:** The permanent logical file structure in which each record's logical position is uniquely identified by a relative record number.

4.164 **relative record number:** The ordinal number of a record in a file whose organization is relative.

4.165 **report:** A printed output described in the report section and generated from those data descriptions.

4.166 **report writer:** A comprehensive set of data clauses and statements that enable a print layout to be described according to its general appearance rather than through of a series of procedural steps.

4.167 **restricted data pointer**: A pointer data item that is restricted to data items of a specified type or to the predefined address NULL.

4.168 **restricted pointer**: A pointer data item that is restricted to data items of a specified type or to programs with the same signature as a specified program.

4.169 **restricted program pointer**: A pointer data item that is restricted to programs with the same signature as a specified program or to the predefined address NULL.

4.170 **run unit:** One or more runtime modules that interact and that function at execution time as an independent entity.

4.171 **runtime element:** The code and data produced by the compilation of a source element.

4.172 **runtime module:** One or more runtime elements that result from the compilation of a compilation unit.

4.173 **sequential access:** An access method in which logical records are either placed into a file in the order of execution of the statements writing the records or obtained from a file in the sequence in which the records were written to the file.

4.174 **sequential organization:** The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

4.175 **sharing mode:** The state of a file that indicates the mode of file sharing.

4.176 **source element:** A source unit excluding any contained source units.

4.177 **source unit:** A sequence of statements beginning with an identification division and finishing with an end marker or the end of the compilation group, including any contained source units.

4.178 **standard arithmetic:** A mode of arithmetic in which the techniques used in handling arithmetic expressions, arithmetic statements, the SUM clause, and certain integer and numeric functions are specified in this final committee draft International Standard.

4.179 **standard intermediate data item:** A temporary abstract decimal floating-point data item used to hold arithmetic operands when standard arithmetic is in effect.

4.180 **static data:** Data that has its last-used state when a runtime element is re-entered.

4.181 **subclass**: A class that inherits from another class. When two classes in an inheritance relationship are considered together, the subclass is the inheritor or inheriting class; the superclass is the inheritee or inherited class.

NOTE In the industry literature, the term derived class is also often used as an alternative to the term subclass. These terms are equivalent.

4.182 **subject of the entry:** The data item that is being defined by a data description entry.

4.183 **subscript:** A number used to refer to a specific element of a table, or in the case of the subscript 'ALL', to all elements of a table.

4.184 **superclass:** A class that is inherited by another class.

4.185 **surrogate pair:** A coded character representation for a single abstract character of the UTF-16 format of the UCS where the representation consists of a sequence of two two-octet values. The first value of the pair is a high-surrogate and the second is a low-surrogate.

4.186 **type (for type declaration):** A template that contains all the characteristics of a data item and its subordinates.

4.187 **UCS; Universal Multiple-Octet Coded Character set:** See Universal Multiple-Octet Coded Character Set.

4.188 **Universal Multiple-Octet Coded Character Set; UCS:** The coded character set defined by ISO/IEC 10646. This coded character set includes the characters used in writing most of the languages of the modern world.

4.189 **universal object reference:** An object reference that is not restricted to a specific class or interface.

4.190 **unsuccessful execution:** The attempted execution of a statement that does not result in the execution of all the operations specified by that statement.

4.191 **zero-length item:** An item whose minimum permitted length is zero and whose length at runtime is zero.

4.192 **zero-length literal:** An alphanumeric or national literal that contains zero characters.

5 Description techniques

5.1 General

The techniques used to describe standard COBOL are:

- General formats
- Rules
- Arithmetic expressions
- Informal description

5.2 General formats

General formats specify the syntax of the elements of standard COBOL and the sequence of arrangement of those elements.

The words, phrases, clauses, punctuation, and operands in each general format shall be written in the compilation group in the sequence given in the general format, unless otherwise specified by the rules of that format.

When more than one arrangement exists for a specific language construct, the general format is separated into multiple formats that are numbered and named.

Elements used in depicting general formats are:

- Keywords
- Optional words
- Operands
- Level numbers
- Options
  - Brackets
  - Braces
  - Choice indicators
- Ellipses
- Punctuation
- Special characters
- Meta-terms that refer to other formats

5.2.1 Keywords

Keywords are reserved words or context-sensitive words. They are shown in uppercase and underlined in general formats. They are required in order to select the functionality associated with that keyword, subject to the conventions specified in 5.2.5, Options, and syntax rules specified for the general format.

5.2.2 Optional words

Optional words are reserved words or context-sensitive words. They are shown in uppercase and not underlined in general formats. They may be written to add clarity when the clause or phrase in which they are defined is written in the source unit.

5.2.3 Operands

An operand is an expression, a literal, or a reference to data or an exception condition. Operands are shown in lowercase and represent values or identification of items, conditions, or objects that the programmer supplies when writing the source unit.

Any term listed below refers to an instance of the corresponding element as described in the text referenced under the column labeled 'described in'. Such instances of the term are represented in lower-case and suffixed with a number (n = 1, 2, ...) for unique reference.

Operand type	Described in	Term (n = 1, 2, 3, ...)
Argument	15.3, Arguments	argument-n
Expression	8.8, Expressions	arithmetic-expression-n boolean-expression-n conditional-expression-n
Integer	5.5, Integer operands	integer-n
Literal	8.3.1.2, Literals	literal-n
Reference	8.4, References	
a) User-defined word, including qualification and subscription if needed	8.3.1.1.1, User-defined words 8.4.1.1, Qualification 8.4.1.2, Subscripts	Any of the types listed in 8.3.1.1.1 suffixed by -n
b) Identifier	8.4.2.1, Identifier	identifier-n
c) Exception name	14.6.12.1.5, Exception-names and exception conditions	exception-name-1

5.2.4 Level numbers

Specific level numbers appearing in general formats are required to be specified when the formats in which they appear are written in the source unit. Level number forms 1, 2, ..., and 9, may be written as 01, 02, ..., 09, respectively.

5.2.5 Options

Options are indicated in a general format by vertically stacking alternative possibilities within brackets, braces, or choice indicators. An option is selected by specifying one of the possibilities from a stack of alternative possibilities or by specifying a unique combination of possibilities from a series of brackets, braces, or choice indicators.

5.2.5.1 Brackets

Brackets, [ ], enclosing a portion of a general format indicate that the syntax element contained within or one of the alternatives contained within the brackets may be explicitly specified or that portion of the general format may be omitted. No default is implied for the omitted element.

5.2.5.2 Braces

Braces, { }, enclosing a portion of a general format indicate that the syntax element contained within the braces or one of the alternatives contained within the braces shall be explicitly specified or is implicitly selected. If one of the alternatives contains only optional words, that alternative is the default and is selected unless another alternative is explicitly specified.

5.2.5.3 Choice indicators

Choice indicators are a pair of bars, |, that enclose a portion of a general format. When enclosed by braces, one or more of the alternatives contained within the choice indicators shall be specified, but any single alternative shall be specified only once. When enclosed by brackets, zero or more of the alternatives contained within the choice indicators shall be specified, but any single alternative may be specified only once. The pair of bars comprising the choice indicator shall be enclosed either by brackets or braces. The alternatives may be specified in any order.

5.2.6 Ellipses

In the general formats, the ellipsis represents the position at which the user elects repetition of a portion of a format. The portion of the format that may be repeated is determined as follows: given an ellipsis in a format,

scanning right to left, determine the right bracket or right brace delimiter immediately to the left of the ellipsis; continue scanning right to left and determine the logically matching left bracket or left brace delimiter; the ellipsis applies to the portion of the format between the determined pair of delimiters.

NOTE In text other than general formats, the ellipsis (...) shows omission of a word or words when such omission does not impair comprehension. This is the conventional meaning of the ellipsis, and the use becomes apparent in context.

5.2.7 Punctuation

The separators comma and semicolon may be used anywhere the separator space is used in general formats and other syntactic specifications. In the compilation group, these separators are interchangeable.

The separator period, when specified in a general format, is required when that format is used.

5.2.8 Special characters

Special character words and separators that appear in formats, although not underlined, are required when such portions of the formats are used.

5.2.9 Meta-terms

Meta-terms appear in lowercase in general formats and are the names of subsections of general formats. Subsections are specified below the main format and are introduced by the phrase 'where *x* is:', with *x* replaced by the meta-term.

5.3 Rules

Except for intrinsic functions, rules are categorized as syntax rules and general rules. Intrinsic functions have argument rules and returned value rules instead.

5.3.1 Syntax rules

Syntax rules supplement general formats, identify equivalent words, and define or clarify the order in which words or elements may be written to form larger elements such as phrases, clauses, or statements. Syntax rules may also impose restrictions on individual words or elements, relax restrictions implied by words or elements, or define a term that may be used in the remaining syntax rules.

The rules of the PICTURE clause specified in 13.18.39.5, Precedence rules, are syntax rules.

When syntax rules specify that a word is synonymous with, an abbreviation for, or equivalent to another word (or words), those words may be written interchangeably and have the same meaning.

5.3.2 General rules

A general rule defines or clarifies the meaning or relationship of meanings of an element or set of elements. It is used to define or clarify the semantics of the statement and the effect that it has on either execution or compilation, and it may define a term that may be used in the remaining general rules.

The rules of the PICTURE clause specified in 13.18.39.4, Editing rules, are general rules.

5.3.3 Argument rules

Argument rules specify requirements, constraints, or defaults associated with arguments to intrinsic functions.

5.3.4 Returned value rules

Returned value rules specify the semantics of an intrinsic function.

5.4 Arithmetic expressions

Some rules contain arithmetic expressions that specify part or all of the results of the COBOL syntax. In presenting the arithmetic expressions, the following additional notation, or different meaning for notation, is used.

5.4.1 Textually subscripted operands

When an operand is textually subscripted, as operand-j_n, the term 'operand-j' identifies a specific operand and 'n' refers to the nth position or occurrence of operand-j.

NOTE An example is in the returned value rules for 15.69, PRESENT-VALUE function.

5.4.2 Ellipses

Ellipses show that the number of terms and operators is variable.

5.5 Integer operands

1) When the term 'integer-n' (n = 1, 2, ...) is used in a general format and associated rules, it refers to a fixed-point integer literal that shall be unsigned and nonzero unless otherwise specified in the associated rules.

2) When the term 'integer' is used as a constraint for an operand in a syntax rule, then

   a) if that operand is a literal, it shall be an integer literal, as defined in 8.3.1.2.2.1, Fixed-point numeric literals;

   b) if that operand is a data-name or an identifier, it shall reference one of the following:

      1. an integer intrinsic function,

      2. a fixed-point numeric data item, other than an intrinsic function, whose description does not include any digit positions to the right of the radix point.

3) When the term 'integer' is used as a constraint for an operand in a general rule, that operand shall evaluate at runtime as follows:

   a) If native arithmetic is in effect, the implementor shall define when the operand is an integer.

   b) If any mode of standard arithmetic is in effect, the operand shall be equal to a standard intermediate data item whose form corresponds to the form of arithmetic that is in effect and whose content has the unique value zero or whose decimal fixed-point representation contains only zeros to the right of the decimal point.

      NOTE 1 If standard arithmetic is in effect, and the value of the exponent is greater than 31, the value of the standard intermediate data item is an integer. If standard-binary or standard-decimal arithmetic is in effect, and the value of the exponent is greater than 34, the value (but not necessarily the form) of the standard intermediate data item is an integer.

      NOTE 2 If the value of the exponent is less than 1 then the value of the standard intermediate data item is not an integer.

   c) If native arithmetic is in effect, the implementor shall define when an arithmetic expression is an integer.

5.6 Informal description

Substantial parts of the COBOL specification are described informally in text, tables, and diagrams other than general format diagrams. These parts normally specify semantics as described in 5.3.2, General rules, but may also include syntactical requirements in addition to those described in 5.2, General formats, and 5.3.1, Syntax rules.

Syntactical requirements are distinguished from semantics by their characteristic of specifying rules for writing source code, as opposed to behavior.

5.7 Hyphens in text

A hyphen appearing at the end of a line of text is part of the character-string or word it divides. Hyphens are not added to divide character-strings or words across lines.

5.8 Verbal forms for the expression of provisions

Specific verbal forms are used in the normative clauses of this final committee draft International Standard in order to distinguish among requirements for compliance, provisions allowing a freedom of choice, and recommendations. Those verbal forms are prescribed by ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards.

Table 1, Verbal forms, summarizes the prescribed verbal forms and equivalent expressions used in this final committee draft International Standard.

Table 1 — Verbal forms

Provision	Verbal form	Alternative expressions
a requirement on an implementation or a program, strictly to be followed for compliance to the standard	shall shall not	is required to is, is to is not permitted is not allowed
a permission expressed by the standard	may need not	is permitted is allowed is not required to
a recommendation expressed by the standard; it need not be followed	should should not	it is recommended that is recommended
a capability or possibility open to the user of the standard	can cannot	is able to it is possible to is possible
Alternative expressions are sometimes used to express other than a requirement; in such cases, the meaning is evident from context.		

6 Reference format

6.1 General

Reference format specifies the conventions for writing COBOL source text and library text. COBOL provides two reference formats: free-form reference format and fixed-form reference format. The two types of reference format may be mixed within and between source text and library text by use of SOURCE FORMAT compiler directives. The default reference format of a compilation group is fixed-form.

The following rules apply to the indicated reference formats:

1) Free-form and fixed-form

   a) Reference format is described in terms of character positions on a line on an input-output medium.

   b) A COBOL compiler shall accept source text and library text written in reference format.

   c) The implementor shall specify the meaning of lines and character positions.

   d) For purposes of analyzing the text of a compilation group, the first character-string of a compilation group is treated as though it were preceded by a separator space and the last character-string of a compilation group is treated as though it were followed by a separator space.

2) Fixed-form

   a) A COBOL compiler shall process fixed-form reference format lines as though the lines had been logically converted from fixed form to free form as described in 6.5, Logical conversion.

   b) After logical conversion, the equivalent free-form lines shall meet the requirements of free-form reference format, except that lines may be longer and all characters of the computer's coded character set shall be retained in alphanumeric and national literals. (See rule 3b.)

3) Free-form

   a) The number of character positions on a line may vary from line to line, ranging from a minimum of 0 to a maximum of 255.

   b) The implementor shall specify any control characters that terminate a free-form line, and whether such control characters may be specified in comments and the content of alphanumeric and national literals.

6.2 Indicators

Indicators are instructions to the compiler for interpreting reference format. Each indicator is classified as either a fixed indicator or a floating indicator.

6.2.1 Fixed indicators

Fixed indicators may be specified in the indicator area of fixed-form reference format as described in 6.3, Fixed-form reference format. The following are fixed indicators:

Character	Indicator name	Indicates
*	comment indicator	a comment line
/	comment indicator	a comment line with page ejection
- (hyphen)	continuation indicator	a continuation line

space source indicator any line that is not a comment line or a continuation line

Fixed indicators are characters in the implementor-defined coded character set or sets used for fixed-form reference format, and are not COBOL characters from the COBOL character repertoire.

NOTE This is significant in that equivalence of alphanumeric and national characters is not required for fixed indicators, nor is it precluded.

6.2.2 Floating indicators

Floating indicators may be used in fixed-form or free-form reference format. The following COBOL character-strings are floating indicators:

Character-string	Indicator name	Indicates
*>	comment indicator	1) a comment line when specified as the first character-string in the program-text area; 2) an inline comment when specified following one or more character-strings in the program-text area, subject to the additional rules in 6.2.2.1, Syntax Rules.
>>	compiler directive indicator	a compiler-directive line when followed by a compiler-directive word - with or without an intervening space, subject to additional rules in 7.3, Compiler directives.
"- '-	literal continuation indicator	continuation of a literal when specified in an unterminated literal with the same quotation symbol in its opening delimiter, subject to additional rules in 6.2.2.1, Syntax Rules.

6.2.2.1 Syntax Rules

1) For purposes of analyzing the text of a compilation group, a space is implied immediately following a floating comment indicator.

2) The floating comment indicator of an inline comment shall be preceded by a separator space, and may be specified wherever a separator space may be specified, except:

   a) as the separator space preceding a floating comment indicator

   b) following a floating literal continuation indicator.

3) All the characters forming a multiple-character floating indicator shall be specified on the same line.

4) A floating literal continuation indicator shall be specified only for an alphanumeric, boolean, or national literal. A given literal shall not be continued with more than one form of continuation.

5) A floating literal continuation indicator shall not be specified on a line that contains a fixed literal continuation indicator.

6) For a continued alphanumeric, boolean, or national literal, the first nonblank character of each continuation line shall be the quotation symbol used in the opening delimiter of the literal.

6.3 Fixed-form reference format

The format of a fixed-form reference format line is depicted in Figure 1 — Fixed-form reference format.

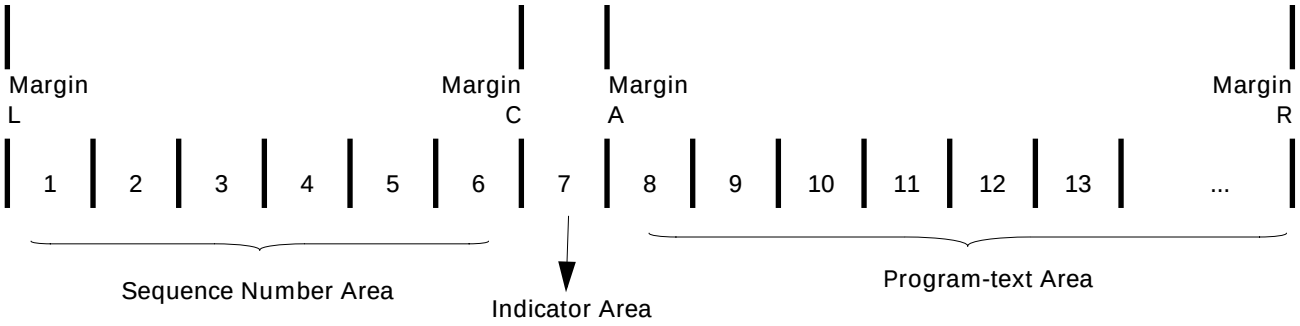


Figure 1 — Fixed-form reference format

Margin L is immediately to the left of the leftmost character position of a line.

Margin C is between the 6th and 7th character positions of a line.

Margin A is between the 7th and 8th character positions of a line.

Margin R is immediately to the right of the rightmost character position of the program-text area. The rightmost character position of the program-text area is a fixed position defined by the implementor.

The sequence number occupies six character positions (1-6), and is between margin L and margin C.

The indicator area is the 7th character position of a line.

The program-text area begins in character position 8 and terminates with the character position immediately to the left of margin R.

6.3.1 Sequence number area

The sequence number area may be used to label a line of source text or library text. The content of the sequence number area is defined by the user and may consist of any character in the computer's coded character set. There is no requirement that the content of the sequence number area appear in any particular sequence or be unique.

6.3.2 Indicator area

The indicator area identifies the type of a source line in accordance with the indicators specified in 6.2.1, Fixed indicators.

6.3.3 Program-text area

The program-text area may contain:

1) Comment-text of a comment line when the indicator area contains a fixed comment indicator.

2) Any of the following or combinations of the following, subject to further syntax specifications, when the indicator area contains a continuation indicator or a source indicator:

   — character-strings (COBOL words)
   — separators

— comments
— floating indicators.

3) All spaces.

6.3.4 Continuation of lines

Any entry, sentence, statement, clause, phrase, or pseudo-text consisting of more than one character-string may be continued by starting subsequent COBOL words, literals, or picture character-strings in the program-text area of a subsequent line.

A COBOL word, literal, or picture character-string may be broken such that part of it appears on one or more subsequent lines. The subsequent lines are continuation lines; the line preceding the continuation line is a continued line. The continuation of a COBOL word, picture character-string, or literal other than an alphanumeric, boolean, or national literal is identified by a fixed continuation indicator in the continuation line.

NOTE Continuation of COBOL words is an archaic feature and its use should be avoided. It is allowed in fixed form for compatibility with the previous COBOL standard.

Continuation of an alphanumeric, boolean, or national literal is indicated when either:

1) A line terminates within an alphanumeric or boolean literal without a closing delimiter and the next line that is not a comment line or a blank line contains a fixed continuation indicator; or

2) A line terminates within an alphanumeric, boolean, or national literal that ends with a floating literal continuation indicator.

In the case of continuation with a fixed continuation indicator, any spaces at the end of the fixed-form continued line are part of the literal.

In the case of continuation with either a fixed or floating literal continuation indicator, the next line that is not a comment line or a blank line is the continuation line. The first nonspace character in the program-text area of the continuation line shall be a quotation symbol matching the quotation symbol used in the opening delimiter. The continuation starts with the character immediately after the quotation symbol in the continuation line.

National literals may be continued only with a floating literal continuation indicator.

All characters composing any multiple-character separator or multiple-character indicator shall be specified on the same line. All characters forming an invocation operator shall be specified on the same line.

Comment lines and blank lines may be interspersed among lines containing the parts of a literal.

If there is no fixed continuation indicator in a line, a space is implied before the first nonblank character in the line for purposes of analyzing the text of the compilation group.

6.3.5 Blank lines

A blank line is one that contains only space characters between margin C and margin R. A blank line may be written as any line of a compilation group.

6.3.6 Comments

A comment consists of a comment indicator followed by comment-text. Any combination of characters from the compile-time computer's coded character set may be included in comment-text.

Comments serve only as documentation and have no effect on the meaning of the compilation group.

A comment may be a comment line or an inline comment.

6.3.6.1 Comment lines

A comment line is identified by either a fixed comment indicator or a floating comment indicator. All characters following the comment indicator up to margin R are comment-text. A comment line may be written as any line of a compilation group.

If a source listing is being produced, a comment line identified by the fixed comment indicator slant (/) causes page ejection followed by printing of the comment line; comments identified by the fixed comment indicator asterisk (*) are printed at the next available line position of the listing.

6.3.6.2 Inline comments

A floating comment indicator preceded by one or more character-strings in the program-text area identifies an inline comment. All characters following the floating comment indicator up to margin R are comment-text. An inline comment may be written on any line of a compilation group except on a line that contains a floating literal continuation indicator.

6.4 Free-form reference format

In free-form reference format, the source or library text may be written anywhere on a line, except that there are specific rules for comments and continuation.

The indicators specified in 6.2.2, Floating indicators, identify specific elements of a compilation group. The entire free-form line constitutes the program-text area of the line.

6.4.1 Continuation of lines

Any entry, sentence, statement, clause, phrase, or pseudo-text consisting of more than one character-string may be continued by writing some of the character-strings and separators that constitute it on successive lines. The last nonblank character of each line is treated as if it were followed by a space.

Alphanumeric literals, boolean literals, and national literals may be continued across lines. The line being continued is called the continued line; subsequent lines are called continuation lines. When such a literal is incomplete at the end of a line, the incomplete portion of the literal shall be terminated by a floating continuation indicator, as defined in 6.2.2, Floating indicators. The continuation indicator may optionally be followed by one or more spaces. The first nonblank character on the continuation line shall be a quotation symbol matching the quotation symbol used in the opening delimiter; the first character after the quotation symbol is the beginning character of the continuation of the literal. At least one alphanumeric character, national character, or hexadecimal digit of the literal content shall be specified on the continued line and on each continuation line.

All characters composing any multiple-character separator or multiple-character indicator shall be specified on the same line. A pair of quotation symbols indicating a single quotation symbol within a literal shall be specified on the same line.

Comment lines and blank lines may be interspersed among lines containing the parts of a literal.

6.4.2 Blank lines

A blank line is one that contains nothing but space characters or is a line with zero character positions. A blank line may appear anywhere in a compilation group.

6.4.3 Comments

A comment consists of a comment indicator followed by comment-text. All characters following the comment indicator up to the end of the line are comment-text.

Any combination of characters from the compile-time computer's coded character set may be included in comment-text, except as indicated in 6, Reference format, rule 3b.

Comments serve only as documentation and have no effect on the meaning of the compilation group.

A comment may be a comment line or an inline comment.

6.4.3.1 Comment lines

A comment line is identified by a floating comment indicator as the first character-string on a line. A comment line may be written as any line in a compilation group.

6.4.3.2 Inline comments

A floating comment indicator preceded by one or more character-strings on a line identifies an inline comment. An inline comment may be written on any line of a compilation group except on a line that contains a floating literal continuation indicator.

6.5 Logical conversion

Source text and library text in fixed-form reference format are logically converted to free-form reference format before the application of replacing and conditional compilation actions. Continued and continuation lines in fixed format and in free format are concatenated to remove continuation indicators, and comments and blank lines are removed from both formats. There is no restriction on the maximum line length of the free-format text resulting from logical conversion.

> NOTE Fixed-form reference format is logically converted during compilation to free-form to simplify understanding of other rules of the language, for example, the COPY statement with the REPLACING phrase and the REPLACE statement. No implementor is required to perform an actual conversion as long as the effect is as though it were performed. The rules of reference format and text manipulation apply regardless of whether there is or is not an actual conversion.

The rules of logical conversion are applied to each line of a compilation group in the order that lines of source text and library text are obtained by the compiler. Lines are examined sequentially beginning with the first line of the compilation group and continuing until the end of the compilation group is reached. The resultant logically-converted compilation group is created in free-form reference format as follows:

1) If the line is a SOURCE FORMAT directive, the reference format mode is determined and the SOURCE FORMAT directive line is logically discarded.

2) If the line is a comment line or a blank line, that line is logically discarded.

3) If the line contains an inline comment, the inline comment is replaced by spaces and processing of that line continues.

4) If the line is a fixed-form or free-form line that contains a floating literal continuation indicator, the end of the program text area is set to immediately follow the character preceding the continuation indicator. The continuation indicator and any following characters are logically discarded, and processing of that line continues.

5) If the line is a fixed-form line, contains a source indicator, and is not a continuation line, the program text area of that line is copied to the resultant compilation group.

6) If the line is a fixed-form continuation line identified by a fixed continuation indicator:

   a) if the continued string is an alphanumeric or boolean literal, the content of the program-text area, beginning with the first character after the initial quotation symbol, is appended immediately to the right of the last character in the latest logical line of the resultant compilation group.

   b) otherwise, the content of the program-text area, beginning with the first nonspace character, is appended immediately to the right of the last character in the latest logical line of the resultant compilation group.

7) If the line is a free-form line and is not a continuation line, that line is copied to the resultant compilation group.

8) If the line is a fixed-form or free-form continuation line that follows a line continued with a floating literal continuation indicator, the content of the program-text area, beginning with the first character after the initial quotation symbol, is appended immediately to the right of the last character in the latest logical line of the resultant compilation group.

9) The next input line is obtained and processing iterates at step 1.

At the end of the compilation group, processing continues with the resultant logically-converted compilation group. The implementor shall define the effect on the source listing, if any, of logical conversion.

7 Compiler directing facility

7.1 General

The compiler directing facility consists of compiler directing statements for text manipulation, compiler directives for text manipulation, and compiler directives for specifying compilation options.

The actions of compiler directing statements and compiler directives occur in two logical stages of compilation group processing - the text manipulation stage and the compilation stage.

The text manipulation stage accepts an initial compilation group, performs modifications specified by COPY and REPLACE statements and conditional compilation directives, and substitutes compilation variables into constant entries. The result is a structured compilation group for processing by the compilation stage.

The compilation stage completes the compilation process utilizing the structured compilation group.

The following are the compiler directing statements and compiler directives and the stage during which their actions take place:

Compiler directing statements	Stage
COPY statement	Text manipulation
SUPPRESS phrase	Implementor-defined
REPLACE statement	Text manipulation

Compiler directives	Stage
CALL-CONVENTION	Compilation
DEFINE	Text manipulation
EVALUATE	Text-manipulation
FLAG-02	Compilation
FLAG-85	Compilation
FLAG-NATIVE-ARITHMETIC	Compilation
IF	Text-manipulation
LEAP-SECOND	Compilation
LISTING	Implementor-defined
PAGE	Implementor-defined
PROPAGATE	Compilation
SOURCE FORMAT	Text-manipulation
TURN	Compilation

The implementor defines the stage during which actions associated with listings, if any, take place.

The substitution of compilation-variable values into constant entries occurs in the text manipulation stage. The manner and time of expansion of parameterized classes and parameterized interfaces is defined by the implementor, except that it occurs after the text manipulation stage of processing.

7.2 Text manipulation

The text manipulation stage of compilation group processing accepts source lines from source text and library text, selectively includes source lines through conditional compilation, and modifies text to produce a structured compilation group.

The following elements and the separators required to distinguish them shall be syntactically correct in the initial source text and library text:

— COPY statements
— compiler directives

— alphanumeric, boolean, and national literals
— fixed and floating indicators
— constant entries specifying a FROM phrase

REPLACE statements shall be syntactically correct after the action of the replacing phrase of the COPY statement.

Other indicators, language elements, and separators need not be syntactically correct until the completion of the text manipulation stage.

Text manipulation consists of processes acting on the lines of source text and library text such that the processes take effect in a specific order. An implementor may optimize the actual processing and interactions in any manner as long as the final result is the same. The following processes are applied in order:

Step 1: An expanded compilation group is created in logical free-form reference format — input lines are accepted sequentially; logically converted to free-form reference format as specified in 6.5, Logical conversion; and placed in the expanded compilation group. Library text identified in COPY statements is incorporated; replacing actions associated with the REPLACING phrase of the COPY statement are deferred to processes associated with step 2. COPY statements and their incorporated text shall be identifiable in the expanded compilation group for purposes of any logically subsequent processing associated with a REPLACING phrase.

Lines that appear in the false path of an IF or EVALUATE directive, including library text identified in COPY statements, may be omitted from the expanded compilation group. SOURCE FORMAT directives in the false path shall be processed to correctly interpret input lines.

NOTE Recognition of true and false paths during logical conversion is neither required nor precluded.

The resulting lines constitute an expanded compilation group.

Step 2: A conditionally-processed compilation group is created — the expanded compilation group is read and the following compiler directives and substitutions are processed in the order encountered in the expanded compilation group:

a) DEFINE, IF, and EVALUATE directives

b) substitution of compilation-variable values into constant entries

c) the replacing actions of COPY statements.

The resulting lines constitute a conditionally-processed compilation group.

Step 3: A structured compilation group is created — the conditionally-processed compilation group is read and the replacing actions of REPLACE statements are applied in order.

The resulting lines constitute a structured compilation group.

References to a compilation group after text manipulation processing are to the structured compilation group, which contains the lines to be used in the compilation stage.

7.2.1 Text manipulation elements

Language elements referenced and not defined in 7, Compiler directing facility, have the meaning defined in 8, Language fundamentals.

7.2.1.1 Compiler directing statements

The compiler directing statements are the COPY statement and the REPLACE statement.

7.2.1.2 Source text and library text

Source text is the primary input to the compiler for a single compilation group. Library text is secondary input to the compiler as a result of processing a COPY statement.

The source text and library text processed by text manipulation consists of indicators, character-strings, comments, and separators. A character-string is either a text-word or the word 'COPY'.

7.2.1.3 Pseudo-text

Pseudo-text is an operand in the REPLACE statement and in the REPLACING phrase of the COPY statement. Pseudo-text may be any sequence of zero or more text-words, comments, and the separator space bounded by, but not including, pseudo-text delimiters.

The opening pseudo-text delimiter and the closing pseudo-text delimiter consist of the two contiguous COBOL characters '=='.

7.2.1.4 Text-words

A text-word is a character-string, other than a comment or the COBOL word 'COPY', in source text or in library text that constitutes an element processed by text manipulation. A text-word may be one of the following:

1) a separator, except for: a space; a pseudo-text delimiter; and the opening and closing delimiters for alphanumeric, boolean, and national literals. In determining which character sequences form text-words, the colon, the right parenthesis, and the left parenthesis characters, in any context except within alphanumeric or national literals, are treated as separators;

2) an alphanumeric, boolean, or national literal including the opening and closing delimiters that bound the literal;

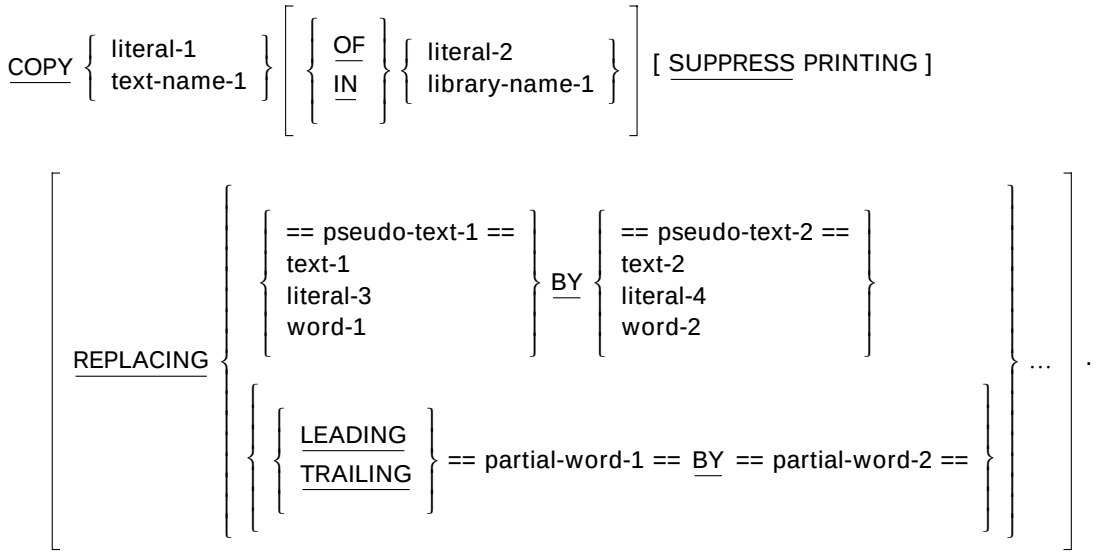
3) any other character or sequence of contiguous characters from the compile-time coded character set, bounded by separators. The implementor may prohibit the use of one or more characters from outside the COBOL character repertoire in, or as, text-words.

   NOTE The contexts in which characters from outside the COBOL character repertoire may be used in elements of COBOL syntax are very limited. Because the syntactic validity of elements or constructs is determined after the completion of all text manipulation, the introduction of non-COBOL characters into such elements or constructs through the action of COPY and REPLACE may have unexpected or undesirable results.

7.2.2 COPY statement

The COPY statement incorporates library text into a COBOL compilation group.

7.2.2.1 General format



7.2.2.2 Syntax rules

1) A COPY statement may be specified anywhere in source text or in library text that a character-string or a separator, other than the closing delimiter of a literal, may appear except that a COPY statement shall not appear within a COPY statement.

2) A COPY statement shall be preceded by a space except when it is the first statement in a compilation group.

3) Within one COBOL library, each text-name shall be unique.

4) A concatenation expression or figurative constant shall not be specified for literal-1, literal-2, literal-3, or literal-4.

5) Literal-1 and literal-2 shall be alphanumeric literals. The allowable value of literal-1 and literal-2 is defined by the implementor.

6) Pseudo-text-1 shall contain one or more text-words, at least one of which shall be neither a separator comma nor a separator semicolon.

7) Pseudo-text-2 shall contain zero, one, or more text-words.

8) Character-strings within pseudo-text-1 and pseudo-text-2 may be continued in accordance with the rules of reference format.

9) Text-1 and text-2 shall be one of the following formats of identifiers: function-identifier, qualified-data-name-with-subscripts, reference-modification, qualified-linage-counter, or qualified-report-counter.

NOTE Text-1 and text-2 are archaic features and their use should be avoided.

If subscripting is specified, it shall not include any arithmetic expressions as subscripts with the exception of a single literal or identifier, or an identifier plus or minus an integer. The format of any identifier specified in subscripts, reference modifiers, or function arguments shall be: a function-identifier, qualified-data-name-with-subscripts, reference-modification, qualified-linage-counter, or qualified-report-counter. Function identifiers in text-1 or text-2 shall be intrinsic function references only.

10) Word-1 or word-2 may be any single COBOL word except 'COPY', the compiler directive indicator, or the comment indicator.

11) The length of a text-word within pseudo-text and within library text shall be from 1 through 65,535 character positions.

12) Compiler directive lines shall not be specified within pseudo-text-1, pseudo-text-2, partial-word-1, or partial-word-2.

13) Partial-word-1 shall consist of one text-word.

14) Partial-word-2 shall consist of zero or one text-word.

15) An alphanumeric, boolean, or national literal shall not be specified as partial-word-1 or partial-word-2.

7.2.2.3 General rules

1) Text-name-1 or literal-1 identifies the library text to be processed by the COPY statement.

2) Library-name-1 names a resource that shall be available to the compiler and shall provide access to the library text referenced by text name-1.

3) The implementor shall define the rules for locating the library text referenced by text-name-1 or literal-1. When neither library-name-1 nor literal-2 is specified, a default COBOL library is used. The implementor defines the mechanism for identifying the default COBOL library.

4) If the SUPPRESS phrase is specified, library text incorporated as a result of COPY statement processing is not listed. If a listing is being produced, the COPY statement itself is listed.

5) At the completion of copying the library text into the compilation group, the LISTING directive that is in effect for the COPY statement itself is considered to be in effect, regardless of any LISTING directives in the library text.

6) The effect of processing a COPY statement is that the library text associated with text-name-1 or the value of literal-1 is copied into the compilation group, logically replacing the entire COPY statement beginning with the reserved word COPY and ending with the separator period, inclusive.

7) If the REPLACING phrase is not specified, the library text is included in the resultant text unchanged.

8) If the REPLACING phrase is specified, library text is modified during creation of the structured compilation group that is described in 7.2, Text manipulation. Each matched occurrence of pseudo-text-1, text-1, word-1, literal-3, or partial-word-1 in the library text is replaced by the corresponding pseudo-text-2, text-2, word-2, literal-4, or partial-word-2 in accordance with subsequent rules of the COPY statement.

9) For purposes of matching, text-1, word-1, and literal-3 are treated as pseudo-text containing only text-1, word-1, or literal-3, respectively.

10) The comparison operation to determine text replacement occurs in the following manner:

    a) The leftmost library text-word that is not a separator comma or a separator semicolon is the first text-word used for comparison. Any text-word or space preceding this text-word is copied into the resultant text. Starting with the first text-word for comparison and first pseudo-text-1, text-1, word-1, literal-3, or

partial-word-1 that was specified in the REPLACING phrase, the entire REPLACING phrase operand that precedes the reserved word BY is compared to an equivalent number of contiguous library text-words.

b) Pseudo-text-1, text-1, word-1, or literal-3 match the library text only if the ordered sequence of text-words that forms pseudo-text-1, text-1, word-1, or literal-3 is equal, character for character, to the ordered sequence of library text-words. When the LEADING phrase is specified, partial-word-1 matches the library text only if the contiguous sequence of characters that forms partial-word-1 is equal, character for character, to an equal number of contiguous characters starting with the leftmost character position of a library text-word. When the TRAILING phrase is specified, partial-word-1 matches the library text only if the contiguous sequence of characters that forms partial-word-1 is equal, character for character, to an equal number of contiguous characters ending with the rightmost character position of a library text-word.

c) The following rules apply for the purpose of matching:

    1. Each occurrence of a separator comma, semicolon, or space in pseudo-text-1 or in the library text is considered to be a single space. Each sequence of one or more space separators is considered to be a single space.

    2. Each operand and operator of a concatenation expression is a separate text-word.

    3. Except when used in the non-hexadecimal formats of alphanumeric and national literals, each alphanumeric character is equivalent to its corresponding national character and each lowercase letter is equivalent to its corresponding uppercase letter, as specified for the COBOL character repertoire in 8.1.2, COBOL character repertoire.

    4. For alphanumeric, boolean and national literals:

        a. The two representations of the quotation symbol match when specified in the opening and closing delimiters of the literal.

            NOTE The opening and closing delimiters are required to be in the same representation.

        b. In the content of the literal, two contiguous occurrences of the character used as the quotation symbol in the opening delimiter are treated as a single occurrence of that character.

    5. Each occurrence of a compiler directive line is treated as a single space.

    6. Comments, if any, are treated as a single space.

        NOTE Because comments are removed during logical conversion, none are expected.

d) If no match occurs, the comparison is repeated with each next successive pseudo-text-1, text-1, word-1, literal-3, or partial-word-1, if any, in the REPLACING phrase until either a match is found or there is no next successive REPLACING operand.

e) When all the REPLACING phrase operands have been compared and no match has occurred, the leftmost library text-word is copied into the resultant text. The next successive library text-word is then considered as the leftmost library text-word, and the comparison cycle starts again with the first pseudo-text-1, text-1, word-1, literal-3, or partial-word-1 specified in the REPLACING phrase.

f) When a match occurs between pseudo-text-1, text-1, word-1, or literal-3 and the library text, the corresponding pseudo-text-2, text-2, word-2, or literal-4 is placed into the resultant text. When a match occurs between partial-word-1 and the library text-word, the library text-word is placed into the resultant text with the matched characters either replaced by partial-word-2 or deleted when partial-word-2 consists of zero text-words. The library text-word immediately following the rightmost text-word that participated in the match is then considered as the leftmost text-word. The comparison cycle starts again with the first pseudo-text-1, text-1, word-1, literal-3, or partial-word-1 specified in the REPLACING phrase.

g) The comparison operation continues until the rightmost text-word in the library text has either participated in a match or been considered as a leftmost library text-word and participated in a complete comparison cycle.

11) If the REPLACING phrase is specified, the library text shall not contain a COPY statement.

12) The resultant text after replacement shall be in logical free-form reference format. When copying text-words into the resultant text, additional spaces may be introduced only between text-words where there already exists a space or where a space is assumed.

NOTE A space is assumed at the end of a source line.

13) If the REPLACING phrase is not specified, the library text may contain a COPY statement that does not include a REPLACING phrase. The implementation shall support nesting of at least 5 levels, including the first COPY statement in the sequence. The library text being copied shall not cause the processing of a COPY statement that directly or indirectly copies itself.

14) The replacing action of a COPY statement shall not introduce a COPY statement, a SOURCE FORMAT directive, a comment, or a blank line.

7.2.3 REPLACE statement

The REPLACE statement modifies text in a compilation group.

7.2.3.1 General format

Format 1 (replacing):

```
REPLACE [ ALSO ] { == pseudo-text-1 == BY == pseudo-text-2 ==
                 { { LEADING  }
                 { { TRAILING } == partial-word-1 == BY == partial-word-2 == } ... .
```

Format 2 (off):

```
REPLACE [ LAST ] OFF .
```

7.2.3.2 Syntax rules

1) A REPLACE statement may be specified anywhere in source text or in library text that a character-string or a separator, other than the closing delimiter of a literal, may appear.

2) A REPLACE statement shall be preceded by a space except when it is the first statement in a compilation group.

3) Pseudo-text-1 shall contain one or more text-words, at least one of which shall be neither a separator comma nor a separator semicolon.

4) Pseudo-text-2 shall contain zero, one, or more text-words.

5) Partial-word-1 shall consist of one text-word.

6) Partial-word-2 shall consist of zero or one text-word.

7) An alphanumeric, boolean, or national literal shall not be specified as partial-word-1 or partial-word-2.

8) Character-strings within pseudo-text-1 and pseudo-text-2 may be continued in accordance with the rules of reference format.

9) The length of a text-word within pseudo-text shall be from 1 through 65,535 characters.

10) Compiler directive lines shall not be specified within pseudo-text-1, pseudo-text-2, partial-word-1, or partial-word-2.

7.2.3.3 General Rules

1) In subsequent general rules of the REPLACE statement, 'source text' refers to the conditionally-processed compilation group.

2) Pseudo-text-1 specifies the text to be replaced by pseudo-text-2.

3) Partial-word-1 specifies the text to be replaced by partial-word-2.

4) Once encountered, a format 1 REPLACE statement has one of three states:

a) active, meaning it is the current statement in use for replace processing for the compilation group;

b) inactive, meaning it is not currently in use for replace processing but is held in a last-in first-out queue, from which it may be popped and made active or canceled in accordance with the rules for subsequent REPLACE statements encountered in the compilation group;

c) canceled, meaning it is removed from use for replace processing for the remainder of the compilation group or, if inactive, it is removed from the queue of inactive statements for the remainder of the compilation group.

5) A REPLACE statement that is placed in the active state remains active until it is placed in the inactive state, it is canceled, or the end of the compilation group is reached, whichever occurs first.

6) When there is no REPLACE statement in the active state:

a) A format 1 REPLACE statement is placed in the active state at the point at which it is encountered in the compilation group. The ALSO phrase, if specified, has no effect.

b) A format 2 REPLACE statement has no effect.

7) When there is a REPLACE statement in the active state:

a) A format 1 REPLACE statement with the ALSO phrase results in the following:

1. the active REPLACE statement is made inactive and is pushed into the queue of inactive REPLACE statements.

2. The current REPLACE statement is expanded into a single REPLACE statement, without the ALSO phrase, having as its operands all the operands of the current statement followed by the operands of the most recent statement pushed into the queue of inactive REPLACE statements. The expanded REPLACE statement is placed in the active state.

b) A format 1 REPLACE statement without the ALSO phrase cancels the active REPLACE statement and cancels any REPLACE statements in the queue of inactive REPLACE statements. Then the current REPLACE statement is placed in the active state.

c) A format 2 REPLACE statement with the LAST phrase cancels the active REPLACE statement and pops the last statement that was pushed into the queue of inactive REPLACE statements, if any. The popped statement, if any, is placed in the active state.

d) A format 2 REPLACE statement without the LAST phrase cancels the active REPLACE statement and cancels all REPLACE statements in the queue of inactive REPLACE statements, if any.

8) The comparison operation to determine text replacement begins with the text immediately following the REPLACE statement and occurs in the following manner:

a) Starting with the leftmost source text-word and the first pseudo-text-1 or partial-word-1, pseudo-text-1 or partial-word-1 is compared to an equivalent number of contiguous source text-words.

b) Pseudo-text-1 matches the source text if, and only if, the ordered sequence of text-words that forms pseudo-text-1 is equal, character for character, to the ordered sequence of source text-words. When the LEADING phrase is specified, partial-word-1 matches the source text-word only if the contiguous sequence of characters that forms partial-word-1 is equal, character for character, to an equal number of contiguous characters starting with the leftmost character position of that source text-word. When the TRAILING phrase is specified, partial-word-1 matches the source text-word only if the contiguous sequence of characters that forms partial-word-1 is equal, character for character, to an equal number of contiguous characters ending with the rightmost character position of that source text-word.

c) The following rules apply for the purpose of matching:

1. Each occurrence of a separator comma, semicolon, or space in pseudo-text-1 or in the source text is considered to be a single space. Each sequence of one or more space separators is considered to be a single space.

2. Each operand and operator of a concatenation expression is a separate text-word.

3. Except when used in the non-hexadecimal formats of alphanumeric and national literals, each alphanumeric character is equivalent to its corresponding national character and each lowercase letter is equivalent to its corresponding uppercase letter, as specified for the COBOL character repertoire in 8.1.2, COBOL character repertoire.

4. For alphanumeric, boolean, and national literals:

   a. The two representations of the quotation symbol match when specified in the opening and closing delimiters of the literal.

      NOTE The opening and closing delimiters are required to be in the same representation.

   b. In the content of the literal, two contiguous occurrences of the character used as the quotation symbol in the opening delimiter are treated as a single occurrence of that character.

5. Each occurrence of a compiler directive line is treated as a single space.

6. Comments, if any, are treated as a single space.

   NOTE Because comments are removed during logical conversion, none are expected.

d) If no match occurs, the comparison is repeated with each next successive occurrence of pseudo-text-1 or partial-word-1, until either a match is found or there is no next successive occurrence of pseudo-text-1 or partial-word-1.

e) When all occurrences of pseudo-text-1 or partial-word-1 have been compared and no match has occurred, the next successive source text-word is then considered as the leftmost source text-word, and the comparison cycle starts again with the first occurrence of pseudo-text-1 or partial-word-1.

f) When a match occurs between pseudo-text-1 and the source text, the corresponding pseudo-text-2 replaces the matched text in the source text. When a match occurs between partial-word-1 and the source text-word, the matched characters of that source text-word are either replaced by partial-word-2 or deleted when partial-word-2 consists of zero text-words. The source text-word immediately following the rightmost text-word that participated in the match is then considered as the leftmost source text-word. The comparison cycle starts again with the first occurrence of pseudo-text-1 or partial-word-1.

g) The comparison operation continues until the rightmost text-word in the source text that is within the scope of the REPLACE statement has either participated in a match or been considered as a leftmost source text-word and participated in a complete comparison cycle.

9) The text produced as a result of processing a REPLACE statement shall not contain a COPY statement, a REPLACE statement, a SOURCE FORMAT directive, a comment, or a blank line.

10) The text that results from the processing of a REPLACE statement shall be in logical free-form reference format. Text-words inserted into the resultant text as a result of processing a REPLACE statement are placed in accordance with the rules of free-form reference format. When inserting text-words of pseudo-text-2 into the resultant text, additional spaces may be introduced only between text-words where there already exists a space or a space is assumed.

    NOTE A space is assumed at the end of a source line.

7.3 Compiler directives

Compiler directives specify options for use by the compiler, define compilation-variables, and control conditional compilation.

7.3.1 General format

>>compiler-instruction

7.3.2 Syntax rules

1) A compiler directive shall be specified on one line, except for the EVALUATE and the IF directives for which specific rules are specified.

2) A compiler directive shall be preceded only by zero, one, or more space characters.

3) When the reference format is fixed-form, a compiler directive shall be written in the program-text area and may be followed only by space characters and an optional inline comment.

4) When the reference format is free-form, a compiler directive may be followed only by space characters and an optional inline comment.

5) A compiler directive is composed of the compiler directive indicator, optionally followed by the COBOL character space, followed by compiler-instruction. The compiler directive indicator shall be treated as though it were followed by a space if no space is specified after the indicator.

6) Compiler-instruction is composed of compiler-directive words, system-names, and user-defined words as specified in the syntax of each directive. Compiler-directive words are identified in 8.12, Compiler-directive words.

7) When a compiler-directive word is specified in the general format of a compiler directive, that compiler-directive word is reserved within the context of that directive.

8) A compiler directive may be specified anywhere in a compilation group, in source text or in library text, except

   a) as restricted by the rules for the specific compiler directive,

   b) within a source text manipulation statement,

   c) between the lines of a continued character-string.

9) The compiler-directive word 'IMP' is reserved for use by the implementor. If the implementor defines the IMP directive, the syntax rules for that directive shall be implementor-defined.

   NOTE >>IMP provides an optional place holder for all current and future implementor-defined directives. In this way the implementor may optionally support the use of >>IMP to indicate the start of one or more implementor-defined directives.

10) A literal in a compiler directive shall not be specified as a concatenation expression, a figurative constant, or a floating-point numeric literal.

7.3.3 General rules

1) A compiler directive line is not affected by the replacing action of a COPY statement or a REPLACE statement.

2) Compiler directives are processed either in the text manipulation stage or the compilation stage of processing, as specified in 7.2, Text manipulation. The order of processing during the text manipulation stage is specified

in 7.2, Text manipulation. During the compilation stage, compiler directives are processed in the order encountered in the structured compilation group.

3) If the implementor defines the IMP directive, the general rules for that directive shall be implementor-defined.

4) A compiler directive applies to all of the source text and library text that follows and is independent of execution flow.

7.3.4 Conditional compilation

The use of certain compiler directives provides a means of including or omitting selected lines of source code. This is called conditional compilation. The compiler directives that are used for conditional compilation are the DEFINE directive, the EVALUATE directive, and the IF directive. The DEFINE directive is used to define compilation variables, which may be referenced in the EVALUATE and IF directives in order to select lines of code that are to be compiled or are to be omitted during compilation. Compilation variables may be referenced in constant entries as specified in 13.10, Constant entry.

7.3.5 Compile-time arithmetic expressions

A compile-time arithmetic expression may be specified in the DEFINE and EVALUATE directives, in a constant conditional expression, and in a constant entry.

7.3.5.1 Syntax rules

1) Compile-time arithmetic expressions shall be formed in accordance with 8.8.1, Arithmetic expressions, with the following exceptions:

   a) The exponentiation operator shall not be specified.

   b) All operands shall be fixed-point numeric literals or arithmetic expressions in which all operands are fixed-point numeric literals.

   c) The expression shall be specified in such a way that a division by zero does not occur and the value of the standard intermediate data item after each operation is within the range specified in 8.8.1.3.1.1, Precision and allowable magnitude.

7.3.5.2 General rules

1) The order of precedence and the rules for evaluation of compile-time arithmetic expressions are shown in 8.8.1, Arithmetic expressions. The results of all arithmetic operations shall be as if ARITHMETIC IS STANDARD-DECIMAL were in effect at compile time.

2) The final result of the arithmetic expression shall be truncated to the integer part of the value as specified in 15.45, INTEGER-PART function, and the resultant value shall be considered to be an integer numeric literal.

7.3.6 Compile-time boolean expressions

A compile-time boolean expression may be specified in the EVALUATE directive and in a constant conditional expression.

7.3.6.1 Syntax rules

1) Compile-time boolean expressions shall be formed in accordance with 8.8.2, Boolean expressions, except that all operands shall be boolean literals or boolean expressions in which all operands are boolean literals.

7.3.6.2 General rules

1) The order of precedence and the rules for evaluation of compile-time boolean expressions are shown in 8.8.2, Boolean expressions.

7.3.7 Constant conditional expression

A constant conditional expression is a conditional expression in which the operands are a defined condition, a literal, or an arithmetic or boolean expression containing only literal terms. A defined condition tests whether a compilation-variable has a defined value.

7.3.7.1 Syntax Rules

1) A constant conditional expression shall be one of the following:

   a) A relation condition in which both operands are literals, arithmetic expressions containing only literal terms, or boolean expressions containing only literal terms. The condition shall be formed according to the rules in 8.8.4.1.1, Relation conditions. The following rules also apply:

      1. The operands shall be of the same category. An arithmetic expression is of the category numeric. A boolean expression is of the category boolean.

      2. If literals are specified and they are not numeric literals, the relational operator shall be 'IS EQUAL TO', 'IS NOT EQUAL TO', 'IS =', 'IS NOT =', or 'IS <>'.

   b) A boolean condition as specified in 8.8.4.1.2, Boolean condition, in which all operands are boolean literals.

   c) A defined condition.

   d) A complex condition as specified in 8.8.4.2, Complex conditions, formed by combining the above forms of simple conditions into complex conditions. Abbreviated combined relation conditions shall not be specified.

2) An arithmetic expression in a constant conditional expression shall be formed in accordance with 7.3.5, Compile-time arithmetic expressions.

3) A boolean expression in a constant conditional expression shall be formed in accordance with 7.3.6, Compile-time boolean expressions.

7.3.7.2 General rules

1) Complex conditions are evaluated according to the rules in 8.8.4.2, Complex conditions.

2) For a simple relation condition where the operands are not numeric or boolean, no collating sequence is used for the comparison. A character by character comparison for equality based on the binary value of each character's encoding is used. If the literals are of unequal length they are not equal.

   NOTE This means that uppercase and lowercase letters are not equivalent.

7.3.7.3 Defined condition

A defined condition tests whether a given compilation-variable is defined.

7.3.7.3.1 General format

compilation-variable-name-1 IS [NOT] DEFINED

7.3.7.3.2 Syntax rules

1) Compilation-variable-name-1 shall not be the same as a compiler-directive word.

7.3.7.3.3 General rules

1) A defined condition using the IS DEFINED syntax evaluates TRUE if compilation-variable-name-1 is currently defined.

2) A defined condition using the IS NOT DEFINED syntax evaluates TRUE if compilation-variable-name-1 is not currently defined.

7.3.8 CALL-CONVENTION directive

The CALL-CONVENTION directive instructs the compiler how to treat references to program-names and method-names and may be used to determine other details for interacting with a function, method, or program.

7.3.8.1 General format

```
>>CALL-CONVENTION { COBOL                 }
                  { call-convention-name-1 }
```

7.3.8.2 General rules

1) The default for the CALL-CONVENTION directive is '>>CALL-CONVENTION COBOL'.

2) The CALL-CONVENTION directive determines how program-names and method-names specified in subsequent INVOKE statements, inline method invocations, CALL statements, CANCEL statements, and program-address-identifiers are processed by the compiler. This directive applies when a program-name or method-name is referenced in those language constructs.

   a) When COBOL is specified, that program-name or method-name is treated as a COBOL word that maps to the externalized name of the method to be invoked or the program to be called, canceled, or referenced in the program-address-identifier, respectively, applying the same implementor-defined mapping rules as for a method-name or program-name for which no AS phrase is specified.

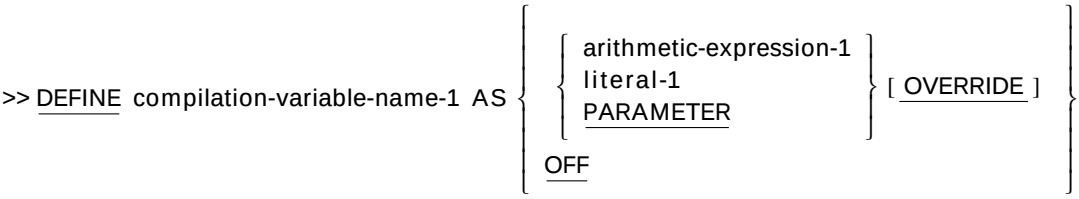
   b) When call-convention-name-1 is specified, that program-name or method-name is treated as a literal that maps to the externalized name of the method to be invoked or the program to be called, canceled, or referenced in the program-address-identifier, respectively, in a manner defined by the implementor.

3) The CALL-CONVENTION directive may also be used by the implementor to determine other details needed to interact with a function, method, or program.

7.3.9 DEFINE directive

The DEFINE directive specifies a symbolic name, called a compilation variable, for a particular literal value. This name may then be used in a constant conditional expression, an EVALUATE directive, or a constant entry. A compilation variable can be set to a value obtained by the compiler from the operating environment.

7.3.9.1 General format



7.3.9.2 Syntax rules

1) Compilation-variable-name-1 shall not be the same as a compiler-directive word.

2) If a DEFINE directive specifies neither the OFF nor the OVERRIDE phrase, then either

   — compilation-variable-name-1 shall not have been declared previously within the same compilation group; or

   — the last previous DEFINE directive referring to compilation-variable-name-1 shall have been specified with the OFF phrase; or

   — the last previous DEFINE directive referring to compilation-variable-name-1 shall have specified the same value.

3) Arithmetic-expression-1 shall be formed in accordance with 7.3.5, Compile-time arithmetic expressions.

7.3.9.3 General rules

1) In text that follows a DEFINE directive specifying compilation-variable-name-1 without the OFF phrase, compilation-variable-name-1 may be used in the compilation group in any compiler directive where a literal of the category associated with the name is permitted, in a defined condition, or in a constant entry where the FROM phrase is specified.

2) Following a DEFINE directive in which the OFF phrase is specified, compilation-variable-name-1 shall not be used except in a defined condition unless it is redefined in a subsequent DEFINE directive.

3) If the OVERRIDE phrase is specified, compilation-variable-name-1 is unconditionally set to reference the value of the specified operand.

4) If the PARAMETER phrase is specified, the value referenced by compilation-variable-name-1 is obtained from the operating environment by an implementor-defined method when the DEFINE directive is processed. If no value is made available from the operating environment, compilation-variable-name-1 is not defined.

5) If the operand of the DEFINE directive consists of a single numeric literal, that operand is treated as a literal, not as an arithmetic-expression.

6) If arithmetic-expression-1 is specified, arithmetic-expression-1 is evaluated according to 7.3.5, Compile-time arithmetic expressions, and compilation-variable-name-1 references the resultant value.

7) If literal-1 is specified, compilation-variable-name-1 references literal-1.

7.3.10 EVALUATE directive

The EVALUATE directive provides for multi-branch conditional compilation.

7.3.10.1 General format

Format 1 (evaluate-value)

```
>> EVALUATE { literal-1 | arithmetic-expression-1 | boolean-expression-1 }

{ >> WHEN { literal-2 | arithmetic-expression-2 | boolean-expression-2 }
    [ { THROUGH | THRU } { literal-3 | arithmetic-expression-3 } ]
    [ text-1 ] } ...

[ >> WHEN OTHER [ text-2 ] ]

>> END-EVALUATE
```

Format 2 (evaluate-truth)

```
>> EVALUATE TRUE

{ >>WHEN constant-conditional-expression-1 [ text-1 ] } ...

[ >> WHEN OTHER [ text-2 ] ]

>> END-EVALUATE
```

7.3.10.2 Syntax rules

ALL FORMATS

1) For descriptive purposes in these syntax rules, operand-1 refers to literal-1, arithmetic-expression-1, or boolean-expression-1 in format 1 and to the TRUE keyword in format 2; operand-2 refers to literal-2, arithmetic-expression-2, or boolean-expression-2 in format 1 and to constant-conditional-expression-1 in format 2; and operand-3 refers to literal-3 or arithmetic-expression-3 in format 1.

2) EVALUATE operand-1 shall begin on a new line and shall be specified entirely on that line.

3) >>WHEN operand-2 [THROUGH operand-3] shall begin on a new line and shall be specified entirely on that line.

4) Text-1 shall begin on a new line.

5) >>WHEN OTHER shall begin on a new line and shall be specified entirely on that line.

6) Text-2 shall begin on a new line.

7) >>END-EVALUATE shall be specified on a new line and shall be specified entirely on that line.

8) Text-1 and text-2 may be any kind of source lines, including compiler directives. Text-1 and text-2 may consist of multiple lines.

9) The phrases of a given EVALUATE directive shall be specified all in the same library text or all in source-text. For purposes of this rule, text-1 and text-2 are not considered phrases of the EVALUATE directive. A nested EVALUATE directive specified in text-1 or in text-2 is considered a new EVALUATE directive.

FORMAT 1

10) Literal-1, arithmetic-expression-1, and boolean-expression-1 are selection subjects. The operands specified in the WHEN phrase are selection objects.

11) All operands of one EVALUATE directive shall be of the same category. For this rule, an arithmetic expression is of category numeric and a boolean expression is of category boolean.

12) If the THROUGH phrase is specified, all selection subjects and selection objects shall be of category numeric.

13) The words THROUGH and THRU are equivalent.

14) Arithmetic-expression-1, arithmetic-expression-2, and arithmetic-expression-3 shall be formed in accordance with 7.3.5, Compile-time arithmetic expressions.

15) Boolean-expression-1 and boolean-expression-2 shall be formed in accordance with 7.3.6, Compile-time boolean expressions.

16) Constant-conditional-expression-1 shall be formed in accordance with 7.3.7, Constant conditional expression.

7.3.10.3 General rules

ALL FORMATS

1) Text-1 and text-2 are not part of the EVALUATE compiler directive line. Any text words in text-1 or text-2 that do not form a compiler directive line are subject to the matching and replacing rules of the COPY statement and the REPLACE statement.

FORMAT 1

2) If an operand of the EVALUATE directive consists of a single numeric literal, that operand is treated as a literal, not as an arithmetic-expression.

3) Boolean-expression-1 and boolean-expression-2 are evaluated in accordance with 7.3.6, Compile-time boolean expressions.

4) The selection subject is compared against the values specified in each WHEN phrase in turn as follows:

   a) If the THROUGH phrase is not specified, a TRUE result is returned if the selection subject is equal to literal-2, arithmetic-expression-2, or boolean-expression-2.

   b) If the THROUGH phrase is specified, a TRUE result is returned if the selection subject lies in the inclusive range determined by literal-2 or arithmetic-expression-2 and literal-3 or arithmetic-expression-3.

   If a WHEN phrase evaluates to TRUE, all lines of text-1 associated with that WHEN phrase are included in the resultant text. All lines of text-1 associated with other WHEN phrases in that EVALUATE directive and all lines of text-2 associated with a WHEN OTHER phrase are omitted from the resultant text.

5) If no WHEN phrase evaluates to TRUE, all lines of text-2 associated with the WHEN OTHER phrase, if specified, are included in the resultant text. All lines of text-1 associated with other WHEN phrases are omitted from the resultant text.

6) If the END-EVALUATE phrase is reached without any WHEN phrase evaluating to TRUE, or without encountering a WHEN OTHER phrase, all lines of text-1 associated with all WHEN phrases are omitted from the resultant text.

7) If literal-1 is an alphanumeric or national literal, a character by character comparison for equality based on the binary value of each character's encoding is used. If the literals are of unequal length they are not equal.

FORMAT 2

8) For each WHEN phrase in turn, the constant-conditional-expression is evaluated in accordance with 7.3.7, Constant conditional expression.

   If a WHEN phrase evaluates to TRUE, all lines of text-1 associated with that WHEN phrase are included in the resultant text. All lines of text-1 associated with other WHEN phrases of that EVALUATE directive and all lines of text-2 associated with a WHEN OTHER phrase are omitted from the resultant text.

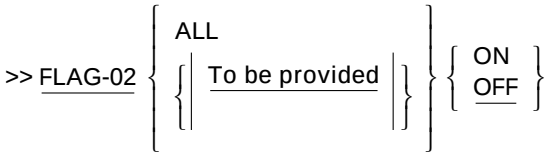
9) If no WHEN phrase evaluates to TRUE, all lines of text-2 associated with the WHEN OTHER phrase, if specified, are included in the resultant text. All lines of text-1 associated with other WHEN phrases are omitted from the resultant text.

10) If the END-EVALUATE phrase is reached without any WHEN phrase evaluating to TRUE, or without encountering a WHEN OTHER phrase, all lines of text-1 associated with all WHEN phrases are omitted from the resultant text.

7.3.11 FLAG-02 directive

The FLAG-02 directive specifies options to flag certain syntax that might be incompatible between the previous COBOL standard and the current COBOL standard.

7.3.11.1 General format



7.3.11.2 Syntax rules

1) The FLAG-02 directive may be specified only between clauses in divisions other than the procedure division and only between statements in the procedure division.

7.3.11.3 General rules

1) The implementor shall provide a warning mechanism that flags the incompatibilities potentially affecting existing programs for the selected option, where the incompatibility is between the specifications in ISO/IEC 1989:2002, and this final committee draft International Standard.

   NOTE A complete list of changes that potentially impact existing programs is given in E.2, Substantive changes potentially affecting existing programs.

2) If ON is explicitly or implicitly specified for an option, the warning mechanism is enabled for that option for all text that follows until the end of the compilation group is reached, a FLAG-02 directive is encountered that turns off all flagging options, or a FLAG-02 directive is encountered that turns off that option.

3) If OFF is specified, flagging for the selected option or options is disabled.

4) The word or words following FLAG-02 indicate the syntax to be diagnosed:

   a) ALL: All of the options apply.

   b) to be provided.

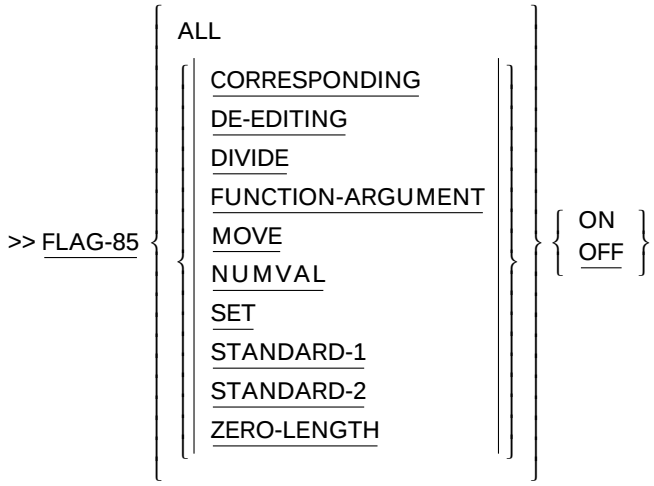
5) If the FLAG-02 directive is not specified, the default for all options is off.

7.3.12 FLAG-85 directive

The FLAG-85 directive specifies options to flag certain syntax for which the behavior might be incompatible between ISO 1989:1985, including its amendments, and ISO/IEC 1989:2002.

NOTE The FLAG-85 directive is an obsolete element in this final committee draft International Standard and is to be deleted from the next edition of standard COBOL.

7.3.12.1 General format



7.3.12.2 Syntax rules

1) The FLAG-85 directive may be specified only between clauses in divisions other than the procedure division and only between statements in the procedure division.

7.3.12.3 General rules

1) The implementor shall provide a warning mechanism that flags the incompatibilities potentially affecting existing programs for the selected options, where the incompatibility is between the specifications in ISO 1989:1985, including Amd 1:1992 and Amd 2:1994, and ISO/IEC 1989:2002.

2) If ON is explicitly or implicitly specified for an option, the warning mechanism is enabled for that option for all text that follows until the end of the compilation group is reached, a FLAG-85 directive is encountered that turns off all flagging options, or a FLAG-85 directive is encountered that turns off that option.

3) If OFF is specified, flagging for the selected option or options is disabled.

4) The word or words following FLAG-85 indicate the syntax to be diagnosed:

   a) ALL: All of the options apply.

   b) CORRESPONDING: In an ADD, MOVE, or SUBTRACT statement with the CORRESPONDING phrase, if subscripting with other than a constant is specified on any of the operands, the statement shall be flagged.

   c) DE-EDITING: A de-editing MOVE statement shall be flagged.

   d) DIVIDE: In a DIVIDE statement with the REMAINDER phrase, if the quotient data item is not described with a sign (there is no S in the PICTURE clause) and either the divisor or dividend is described with a sign, the DIVIDE statement shall be flagged.

e) FUNCTION-ARGUMENT: If the intrinsic function RANDOM is followed immediately by an arithmetic expression that is enclosed in parentheses, and the function is specified as an argument in an intrinsic function that allows multiple arithmetic expressions as arguments, the function RANDOM shall be flagged.

f) MOVE: If an alphanumeric literal or data item is moved to a numeric data item and the number of characters in the sending operand is greater than 31, the MOVE statement shall be flagged.

g) NUMVAL: The NUMVAL and NUMVAL-C intrinsic functions shall be flagged.

h) SET: If a condition-setting format SET statement references a variable length group item, the SET statement shall be flagged.

i) STANDARD-1: if STANDARD-1 is specified in an ALPHABET clause in the SPECIAL-NAMES paragraph, the ALPHABET clause shall be flagged.

j) STANDARD-2: if STANDARD-2 is specified in an ALPHABET clause in the SPECIAL-NAMES paragraph, the ALPHABET clause shall be flagged.

k) ZERO-LENGTH: A READ statement that could return a zero-length item or any statement that references a data item that could be a zero-length item shall be flagged.

5) If the FLAG-85 directive is not specified, the default for all options is off.

7.3.13 FLAG-NATIVE-ARITHMETIC directive

The FLAG-NATIVE-ARITHMETIC directive specifies that when standard arithmetic has been specified or implied for the source unit, any arithmetic operation that uses native arithmetic shall be flagged.

NOTE The FLAG-NATIVE-ARITHMETIC directive is an obsolete element in this final committee draft International Standard and is to be deleted in the next edition of standard COBOL.

7.3.13.1 General format

```
>>FLAG-NATIVE-ARITHMETIC { ON  }
                         { OFF }
```

7.3.13.2 Syntax rules

1) The FLAG-NATIVE-ARITHMETIC directive may be specified only between clauses in divisions other than the procedure division and only between statements in the procedure division.

7.3.13.3 General rules

1) The implementor shall provide a warning mechanism that flags the use of native arithmetic when ARITHMETIC IS STANDARD has been specified or implied for the source unit.

2) If ARITHMETIC IS STANDARD has been specified or implied for the source unit and ON is explicitly or implicitly specified, flagging is enabled for any operation that uses or has the potential to use native arithmetic.

3) If OFF is specified, no flagging for native arithmetic shall occur.

4) If native arithmetic is in effect, no flagging of native arithmetic occurs.

5) If the FLAG-NATIVE-ARITHMETIC directive is not specified, the default is off.

7.3.14 IF directive

The IF directive provides for 1- or 2-way conditional compilation.

7.3.14.1 General format

```
>> IF constant-conditional-expression-1 [ text-1 ]
  [ >> ELSE [ text-2 ] ]
   >> END-IF
```

7.3.14.2 Syntax rules

1) >>IF conditional-expression-1 shall begin on a new line and shall be specified entirely on that line.

2) Text-1 shall begin on a new line.

3) >>ELSE shall begin on a new line and shall be specified entirely on that line.

4) Text-2 shall begin on a new line.

5) >>END-IF shall begin on a new line and shall be specified entirely on that line.

6) Text-1 and text-2 may be any kind of source lines, including compiler directives. Text-1 and text-2 may consist of multiple lines.

7) The phrases of a given IF directive shall be specified all in the same library text or all in source-text. For purposes of this rule, text-1 and text-2 are not considered phrases of the IF directive. A nested IF directive specified in text-1 or in text-2 is considered a new IF directive.

7.3.14.3 General rules

1) Text-1 and text-2 are not part of the IF compiler directive line. Any text words in text-1 or text-2 that do not form a compiler directive line are subject to the matching and replacing rules of the COPY statement and the REPLACE statement.

2) If constant-conditional-expression-1 evaluates to TRUE, all lines of text-1 are included in the resultant text and all lines of text-2 are omitted from the resultant text.

3) If constant-conditional-expression-1 evaluates to FALSE, all lines of text-2 are included in the resultant text and all lines of text-1 are omitted from the resultant text.

7.3.15 LEAP-SECOND directive

The LEAP-SECOND directive specifies whether a value greater than or equal to 60 may be returned in the seconds portion of the value returned by the ACCEPT statement with the TIME phrase, the CURRENT-DATE intrinsic function, the FORMATTED-CURRENT-DATE intrinsic function, and the WHEN-COMPILED intrinsic function. It also specifies whether a value greater than or equal to 86,400 may be returned by the SECONDS-PAST-MIDNIGHT intrinsic function. This directive specifies whether a value in the seconds subfield of a formatted time value may be greater than or equal to 60 and whether a value in standard numeric time form may be greater than or equal to 86,400.

7.3.15.1 General format

```
>>LEAP-SECOND { ON  }
              { OFF }
```

7.3.15.2 Syntax rules

1) The LEAP-SECOND directive shall not be specified within a compilation unit.

7.3.15.3 General rules

1) If the LEAP-SECOND directive is not specified, a LEAP-SECOND directive with the OFF phrase is implied before the first compilation unit in the compilation group.

2) When ON is specified or implied, the implementor defines whether a value greater than 59 may be reported in the seconds position of the value returned from:

   — the ACCEPT statement with the TIME phrase
   — the CURRENT-DATE intrinsic function
   — the FORMATTED-CURRENT-DATE intrinsic function
   — the WHEN-COMPILED intrinsic function.

3) When OFF is specified or implied, a value greater than 59 shall not be reported in the seconds position of the value returned from:

   — the ACCEPT statement with the TIME phrase
   — the CURRENT-DATE intrinsic function
   — the FORMATTED-CURRENT-DATE intrinsic function
   — the WHEN-COMPILED intrinsic function.

4) When ON is specified or implied, a standard numeric time form value shall be greater than or equal to zero and less than 86,401. The implementor defines whether a value greater than or equal to 86,400 may be returned from the SECONDS-PAST-MIDNIGHT intrinsic function.

5) When OFF is specified or implied, a standard numeric time form value shall be greater than or equal to zero and less than 86,400.

6) When ON is specified or implied, the value contained in the seconds subfield of a formatted time value shall be greater than or equal to zero and less than 61.

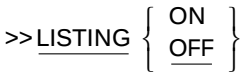
7) When OFF is specified or implied, the value contained in the seconds subfield of a formatted time value shall be greater than or equal to zero and less than 60.

7.3.16 LISTING directive

The LISTING directive instructs the compiler to turn any source listing on or off.

NOTE This final committee draft International Standard does not define the content or layout of any listing. It is recommended that the implementor provide a listing of the original compilation group and, optionally, a listing of the result of any text manipulation applied to the original compilation group.

7.3.16.1 General Format



7.3.16.2 General Rules

1) Whether the compiler produces a source listing is implementor-defined. If the compiler does not produce a source listing, the LISTING directive shall be ignored. Otherwise, the following general rules apply.

2) The default LISTING directive is '>>LISTING ON'.

3) Each LISTING directive shall be listed, even if the listing is being suppressed by a LISTING directive.

4) If OFF is specified, source lines shall not be listed until a LISTING directive specifying or implying the ON phrase is encountered, with the exception that another LISTING OFF directive shall be listed.

5) If ON is specified or implied, source lines shall be listed until either a LISTING OFF directive is encountered or the end of the compilation group is reached.

7.3.17 PAGE directive

The PAGE directive specifies page ejection and provides documentation for the source listing.

7.3.17.1 General format

```
>>PAGE [ comment-text-1 ]
```

7.3.17.2 Syntax rules

1) Comment-text-1 may contain any character in the compile-time computer's coded character set except for control characters as specified in 6, Reference format, rule 3b.

2) Comment-text-1 is not checked syntactically.

7.3.17.3 General rules

1) Comment-text-1 shall serve only as documentation.

2) If a source listing is being produced, a PAGE directive shall cause page ejection followed by listing of the PAGE directive.

3) If a source listing is not being produced, a PAGE directive shall have no effect.

7.3.18 PROPAGATE directive

The PROPAGATE directive is used to cause propagation of exception conditions to the activating runtime element.

7.3.18.1 General format

```
>>PROPAGATE { ON  }
            { OFF }
```

7.3.18.2 Syntax rules

1) A PROPAGATE directive shall not be specified within a compilation unit.

7.3.18.3 General rules

1) When the ON phrase is specified or implied, automatic propagation of exception conditions becomes enabled for functions, methods, and programs that follow in the compilation group. Automatic propagation remains enabled until a PROPAGATE directive specifying the OFF phrase is encountered or the end of the compilation group is reached.

2) During execution of a runtime element for which automatic propagation of exception conditions is enabled, any exception condition raised and not handled by either an exception phrase or a declarative in that runtime element shall be propagated as though a GOBACK RAISING LAST statement were executed in a declarative for that exception condition.

3) When the OFF phrase is specified, automatic propagation of exception conditions becomes disabled for functions, methods, and programs that follow in the compilation group until a PROPAGATE directive specifying the ON phrase is encountered.

4) The default for a compilation group is PROPAGATE OFF.

7.3.19 SOURCE FORMAT directive

The SOURCE FORMAT directive specifies whether the reference format of the source text or library text that follows is fixed form or free form.

7.3.19.1 General format

```
>>SOURCE FORMAT IS { FIXED }
                   { FREE  }
```

7.3.19.2 General rules

1) The SOURCE FORMAT directive indicates that the source text or library text following the directive and continuing through a subsequent SOURCE FORMAT directive shall be treated as fixed form if FIXED is specified, or as free form if FREE is specified. (See 6.3, Fixed-form reference format, and 6.4, Free-form reference format.)

2) The default reference format of a compilation group is fixed form.

3) The default reference format of library text is the reference format that was in effect for the COPY statement that resulted in processing of this library text.

4) A SOURCE FORMAT directive that is the first line of a compilation group or library text may be in either fixed form or free form.

5) If a SOURCE FORMAT directive is specified in library text, the specified format shall be in effect until another SOURCE FORMAT directive is encountered or the end of the library text is reached. When the processing of that library text is completed, the reference format shall revert to the reference format that was in effect for the COPY statement that resulted in processing of that library text.

7.3.20 TURN directive

The TURN directive is used to turn checking for specified exception conditions on or off.

7.3.20.1 General format

```
>>TURN { exception-name-1 [ file-name-1 ] ...} ... CHECKING { ON [ WITH LOCATION ] }
                                                           { OFF                    }
```

7.3.20.2 Syntax rules

1) Any user-defined word beginning with the COBOL characters 'EC-' is interpreted as an exception-name-1 rather than as file-name-1. Any user-defined word that duplicates a compiler-directive word is interpreted as a compiler-directive word rather than as file-name-1.

2) Exception-name-1 shall be one of the exception names listed in 14.6.12.1, Exception conditions. There shall be no verification that a user-defined exception-name or a file-name specified in a TURN directive is actually used within the range of the TURN directive.

   NOTE An exception object is always enabled.

3) No exception-name shall be specified more than once in one TURN directive.

4) No file-name shall be specified more than once for one exception condition.

5) If file-name-1 is specified, exception-name-1 shall begin with the COBOL characters 'EC-I-O'.

7.3.20.3 General rules

1) The default TURN directive is '>>TURN EC-ALL CHECKING OFF'.

2) If exception-name-1 EC-ALL is specified, the effect is as if the same TURN directive were specified containing all exception-names.

3) If exception-name-1 is one of the level-2 exception-names, the effect is as if that TURN directive were specified containing all exception-names that are subordinate to that level-2 exception-name. If file-name-1 is specified, the effect is as if file-name-1 were specified for each of these exception-names.

4) If specified within a statement, the TURN directive does not apply to any phrase of that statement. That TURN directive applies to any succeeding statement in the sequence of source lines, whether or not that succeeding statement is within the scope of the statement in which the TURN directive is specified.

5) If the ON phrase is specified or implied, checking for the exception condition associated with exception-name-1 is enabled for the procedure division statements and procedure division headers that follow in the compilation group; if file-name-1 is specified, checking is enabled only for exception conditions associated with that file-name. Checking remains enabled for every qualifying procedure division statement and procedure division header that follows in the compilation group until it is disabled by a TURN directive with the OFF phrase.

6) If the LOCATION phrase is specified, all information necessary to identify a source statement for the EXCEPTION-LOCATION, EXCEPTION-LOCATION-N, and EXCEPTION-STATEMENT functions is made available to the run unit. If the LOCATION phrase is not specified, the implementor shall specify whether this information is made available or not.

7) If the OFF phrase is specified, checking for the exception condition associated with exception-name-1 is disabled for all procedure division statements and procedure division headers that follow in the compilation

group and remains disabled until another TURN directive for exception-name-1 with the ON phrase is encountered; if file-name-1 is specified, checking is disabled only for exception conditions associated with that file-name.

8 Language fundamentals

8.1 Character sets

The character set concepts in COBOL are the computer's coded character set, the COBOL character repertoire, and alphabets.

The computer's coded character set is the character set used for COBOL's internal processing.

The COBOL character repertoire is a repertoire of characters used in defining the syntax of the language. It is an abstract character set in that it is a list of characters independent of their encoding. The elements of the language that are specified in the COBOL character repertoire are given in 8.1.2, COBOL character repertoire.

Alphabets identify coded character sets for representing data on external media or identify collating sequences, or both. The programmer can define alphabets in the SPECIAL-NAMES paragraph or reference predefined alphabets identified in the SPECIAL-NAMES paragraph.

The CODE-SET clause may be used in a file description entry, referencing alphabets as defined in SPECIAL-NAMES, to describe alternative encoding of records on external media. During input-output operations, records for files declared with the CODE-SET clause are converted to and from that encoding and the encoding of the computer's coded character set.

8.1.1 Computer's coded character set

The computer's coded character set is the set of characters used in the memory of the computer during compilation or during execution of a COBOL runtime element.

In source code, the content of alphanumeric and national literals, except for hexadecimal formats, may contain any characters in the computer's coded character set used for writing source code, consistent with the characters the implementor allows for the class of the literal. The coded character set used during compilation may be the same as or different from the coded character set used during execution of the resultant runtime elements.

In source code, comments may contain any characters in the coded character set that is used for writing source code, subject to the rules in 6, Reference format.

The runtime computer's coded character set consists of a coded character set used to represent data described as usage display and a coded character set used to represent data described as usage national, called the computer's alphanumeric coded character set and the computer's national coded character set, respectively. The alphanumeric coded character set and the national coded character set may be two distinct coded character sets, or they may be one coded character set where a subset is designated as alphanumeric and the set or a subset is designated as national. In either case, unless specifically qualified as alphanumeric or national, the term computer's coded character set references both the alphanumeric and the national coded character sets. The characters of the alphanumeric coded character set and the characters of the national coded character set may, but need not be, disjoint sets.

NOTE 1 In general, the specification assumes that the national character set includes the characters of the alphanumeric character set; for example, intrinsic functions are defined for conversion between the two. An alphanumeric character set is typically a Latin alphabet coded character set, such as ISO/IEC 646, but may be any coded character set. A national character set is intended for larger coded character sets, such as the Universal Multiple-Octet Coded Character Set (UCS) defined by ISO/IEC 10646\, but may be any coded character set.

NOTE 2 An example of one coded character set used to represent both the alphanumeric and the national coded character set is UTF-16 in the UCS where the national coded character set might consist of the entire UCS coded character set and the alphanumeric coded character set might consist of a subset of the UCS. Nothing precludes the alphanumeric and the national coded character sets from both consisting of the entire UCS coded character set.

At runtime, an implementor may recognize a combination of characters from the computer's alphanumeric coded character set and the computer's national coded character set in the content of data items of category

alphanumeric. This combination is referred to as mixed alphanumeric and national data. When this capability is provided, the implementor shall specify any applicable general rules.

The COBOL specification is independent of the encoding used to represent a computer's coded character set, except that:

1) The number of bytes used in the memory of the computer to represent characters in the alphanumeric coded character set shall be the same for all characters in that coded character set; the number of bytes shall be determined at compile time.

2) The number of bytes used in the memory of the computer to represent characters in the national coded character set shall be the same for all characters in that coded character set; the number of bytes shall be determined at compile time.

3) The number of bytes used to represent a character of the national coded character set shall be equal to or greater than the number of bytes used to represent a character of the alphanumeric coded character set.

   NOTE COBOL processes each fixed-size element of a character set as one character, even when a graphic symbol requires two or more elements for its representation in that character set.

Source code rules are described in 8.1.2, COBOL character repertoire.

The implementor shall specify the number of bits in a byte for each supported computer.

The implementor shall specify the set of characters in and the encoding of each of the computer's alphanumeric character set and the computer's national character set. When these are implemented as one character set, the implementor shall specify the characters that map into the computer's alphanumeric coded character set and the characters that map into the computer's national coded character set. If more than one encoding of the computer's character set is supported, the implementor shall specify the mechanism for selecting the encoding for use at runtime.

The implementor shall specify the coded character values associated with the following COBOL items:

- the figurative constants SPACE, QUOTE, and ZERO
- the character used for space filling of alphanumeric data items
- the character used for space filling of national data items
- editing characters
- characters used for numeric digits.

When the computer's coded character set at runtime differs from the coded character set known at compile time, the content of alphanumeric and national literals shall be converted, prior to use at runtime, to the computer's runtime alphanumeric or national coded character set as appropriate for the class of the literal, except that the hexadecimal-alphanumeric format and the hexadecimal-national format literals shall not be converted. The implementor shall define the correspondence of each character of the compile-time coded character set with an associated character in the runtime coded character set. If the runtime coded character set is known at compile time, the conversion may occur either at compile time or at runtime. If the runtime coded character set is not known at compile time, conversion occurs at runtime. The implementor determines the point at which runtime conversion occurs.

When the computer's compile-time coded character set includes characters that are not also included in the COBOL character repertoire, the implementor shall specify any such additional characters that are prohibited from use as a currency symbol.

8.1.2 COBOL character repertoire

The COBOL character repertoire is used to specify the syntax of the language. COBOL words, separators, picture symbols, numeric literals, the currency sign, floating format indicator characters, and the content of boolean and

hexadecimal literals are defined in the COBOL character repertoire. The implementor maps the COBOL character repertoire to one or more coded character sets to be used in writing source code.

The COBOL character repertoire consists of the basic letters, basic digits, basic special characters, and extended letters as shown in table 2, COBOL character repertoire. Extended letters permit writing user-defined words in many languages in addition to the English language.

Table 2 — COBOL character repertoire

Description	Character	Meaning
Basic letters	A, B, ... Z	the basic Latin capital letters in ISO/IEC 646 or in ISO/IEC 10646
	a, b, ... z	the basic Latin small letters in ISO/IEC 646 or in ISO/IEC 10646
Basic digits	0, 1, ... 9	digits
Basic special character		space
	+	plus sign
	–	minus sign (hyphen)
	*	asterisk
	/	slant (slash, solidus)
	=	equal sign
	$	currency sign
	,	comma
	;	semicolon
	.	period
	"	quotation mark
	'	apostrophe
	(	left parenthesis
	)	right parenthesis
	>	greater than
	<	less than
	&	ampersand
	:	colon
	_	underscore
Extended letters		additional characters from the repertoire of ISO/IEC 10646 used in formation of user-defined words

8.1.2.1 General rules

1) The implementor shall define a mapping of the basic letters, basic digits, basic special characters, and extended letters of the COBOL character repertoire to one or more coded character sets. The COBOL character repertoire may be represented in any encoding scheme chosen by the implementor, including but not limited to one coded character set containing all characters of the repertoire or two distinct coded character sets, one alphanumeric and one national, mixed together. When two distinct coded character sets are used, the implementor shall define a correspondence between the basic letters, basic digits, and basic special characters of the alphanumeric and national coded character sets.

NOTE 1 The concepts 'alphanumeric character' and 'national character' apply to the encoding of data. The concepts 'basic letter' and 'extended letter' apply to source code and specify the symbols themselves, not their encoding. A given instance of a basic letter in a compilation group may be encoded in either an alphanumeric coded character set or in a national coded character set, but it is the same letter in either case. For example, within a compilation group, a basic letter 'A' encoded in ISO/IEC 646 has the same meaning in COBOL words as a basic letter 'A' encoded in ISO/IEC 10646 -- just as an uppercase 'A' has the same meaning as a lowercase 'a'.

NOTE 2 Examples of coded character sets that may be used to represent the COBOL character repertoire are ISO/IEC 10646 UCS-4, UTF-8, or UTF-16; and implementor-defined coded character sets consisting of two distinct coded

character sets mixed together, one a national coded character set and one an alphanumeric coded character set. There are other possible implementor-defined encodings of the COBOL character repertoire.

2) If the COBOL character repertoire is mapped to mixed alphanumeric and national coded character sets, the implementor shall specify the control functions or other mechanism for distinguishing alphanumeric characters from national characters. If more than one encoding is permitted in a single compilation group, the implementor shall specify the control functions or other methods used for distinguishing between encodings. Any control functions used to switch between coded character sets are utilized in the compilation process and are not part of the syntax of the compilation group unless otherwise specified by the implementor.

3) Within a compilation group, the following rules apply:

   a) COBOL basic letters, when specified in the non-hexadecimal formats of alphanumeric and national literals, are treated in a case-sensitive manner, unless specific rules require otherwise. COBOL basic letters appearing elsewhere within the compilation group are treated in a case-insensitive manner.

   b) Each basic letter, basic digit, basic special character, and extended letter represented in the alphanumeric character set is equivalent to its corresponding basic letter, basic digit, basic special character, and extended letter, represented in the national character set, respectively.

   Equivalence of uppercase and lowercase basic letters is achieved by folding from uppercase to lowercase in accordance with the case mapping described in Annex C.

4) The set of extended letters consists of characters from the repertoire specified in Annex B, Characters permitted in user-defined words, excluding any character that is defined as a basic letter, basic digit, or basic special character in the COBOL character repertoire. Extended letters in user-defined words are subject to the following rules:

   a) A character in Annex B is included in the set of extended letters if it exists in the implementor-defined compile-time coded character set.

   b) Characters identified as special characters in Annex B shall not be written as the first or last character of a user-defined word. Characters identified as combining characters in Annex B shall not be written as the first or last character of a user-defined word.

   c) An uppercase extended letter is treated as though it were folded to its corresponding lowercase extended letter, if any, in accordance with the case mapping described in Annex C, except when specific rules require uppercase letters and their corresponding lowercase letters to be considered distinct from each other.

      NOTE Extended letters in the COBOL character repertoire are an optional feature in this committee draft International Standard.

   d) If the set of extended letters includes any of the combining characters identified in Annex B, the base character and each combining character are treated as separate characters in determining the length of a user-defined word.

      NOTE 1 For portable source code, programmers need to form user-defined words from the basic letters, the basic digits, the underscore, and the hyphen in the COBOL character repertoire.

      NOTE 2 Annex B identifies characters recommended for use in programming language identifiers. The list of characters in Annex B excludes punctuation and symbols that are not generally used in words or that are considered inappropriate for programming language identifiers. Some characters in Annex B may have an appearance similar to basic special characters specified in COBOL or may appear strange to speakers of some languages, but are necessary for representing certain languages. They are permitted in COBOL on the assumption that no confusion will result for user-defined words written by programmers fluent in the languages for which those characters are essential.

      NOTE 3 The list of characters recommended for user-defined words in Annex B includes combining sequences from ISO/IEC 10646 level 2, but not from level 3. The list does not include combining sequences that form alternate representations of composite characters such as é.

NOTE 4 Extended letters are case folded to lowercase for determining equivalency of uppercase and lowercase.

5) When an implementation does not provide a graphic representation of all characters of the COBOL character repertoire, substitute graphics may be specified by the implementor to replace the characters not represented.

8.1.3 Alphabets

Alphabets in COBOL are named specifications of coded character sets or collating sequences or both. The SPECIAL-NAMES paragraph provides the means for naming alphabets and for specifying user-defined coded character sets and collating sequences. A coded character set or collating sequence is used by specifying its alphabet-name in COBOL statements or entries that reference a coded character set or collating sequence as an operand.

8.1.4 Collating sequences

A collating sequence defines the order of characters within a coded character set or COBOL alphabet for purposes of sorting, merging, and comparing data and for processing files with indexed organization. Logically, there are two collating sequences - an alphanumeric collating sequence and a national collating sequence. An alphanumeric collating sequence defines the order associated with data items or record keys described as usage display; a national collating sequence defines the order associated with data items or record keys described as usage national. These two logical collating sequences may be defined and implemented separately or may be defined and implemented as one composite collating sequence with characters mapped into a logical alphanumeric collating sequence and a logical national collating sequence.

The default ordering associated with these collating sequences is defined by the implementor. Specific orderings may be selected:

— as the program collating sequence, by specification of an alphabet or a locale in the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph;

— for SORT or MERGE statements, by specification of a locale or alphabet in a SORT or MERGE statement;

— for indexed files, by specification of a locale or alphabet in a COLLATING SEQUENCE clause of the file control entry;

— for specific comparisons, by use of the LOCALE-COMPARE or STANDARD-COMPARE intrinsic functions.

When a locale is specified, the associated ordering is determined at runtime.

8.2 Locales

A locale provides a specification of cultural elements for use at runtime. Cultural elements are grouped into named locale-categories that control specific aspects of runtime behavior, as follows:

Locale-category name	Behavior affected
LC_COLLATE	Collating sequence
LC_CTYPE	Character classification and case conversion
LC_MESSAGES	Formats of informative and diagnostic messages and interactive responses
LC_MONETARY	Monetary formatting
LC_NUMERIC	Numeric formatting
LC_TIME	Date and time formats
LC_ALL	Locale-categories LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME, and any other categories included in the locale.

Locale category names, the details of locale-categories, and locale field names are defined in ISO/IEC 9945. The format and implementation of locales may differ from ISO/IEC 9945 provided that logically-equivalent functionality is supported.

When the use of cultural elements from a locale is specified for a source unit, the specific values, formats, or algorithms associated with the locale categories are determined at runtime.

Some operating environments provide a locale for system-wide use, called a system-default locale. Those environments might also provide for selection of a locale for use within a run unit, called a user-default locale.

At the time a run unit is activated, the current runtime locale is set to the user default locale and remains in effect for the run unit until another runtime locale is established. The SET statement provides the capability of establishing any locale as the current runtime locale, as well as the ability to set the user default locale to any locale. While there is always a current locale for the entire run unit, it has effect only for compilation units using language features that reference a locale.

Execution of a SET statement specifying USER-DEFAULT as the sending operand sets the current runtime locale for the specified locale categories to the user default locale. The implementor shall specify the manner in which the user default locale is defined and shall provide at least one user default locale for use in computing environments that do not provide a user default locale.

Execution of a SET statement specifying SYSTEM-DEFAULT as the sending operand sets the current runtime locale for the specified locale categories to the current system default locale. The implementor shall specify the manner in which the system default locale is defined and shall provide at least one system default locale for use in computing environments that do not provide a system default locale.

Execution of a SET statement specifying a locale-name as the sending operand sets the current runtime locale for the specified locale categories to the locale associated with that locale-name in the LOCALE clause of the SPECIAL-NAMES paragraph. The SET statement can be used to save information about the current locale so that the particular locale can later be made current by using another SET statement.

If the user default locale or the system default locale is switched by a non-COBOL runtime module, the new user default or system default locale is not utilized by COBOL unless a SET statement is executed to make it the current runtime locale. A locale switch for any locale categories by an activated COBOL runtime module is utilized on return by the activating runtime module. It is implementor-defined whether, and for which locale categories, a switch of current locale by a non-COBOL runtime module is utilized by COBOL.

NOTE The capability of setting the system default locale from COBOL is not provided.

The manner of identifying the current locale is specified in 14.6.6, Locale identification.

If the locale is not found during an operation requiring a locale, the EC-LOCALE-MISSING exception condition is set to exist and the operation is unsuccessful. If the locale content is invalid or incomplete during an operation using a locale, the EC-LOCALE-INVALID exception condition is set to exist and the operation is unsuccessful.

If a locale does not define both alphanumeric and national collating sequences in category LC_COLLATE, the locale shall define a national collating sequence such that it contains characters to which a correspondence exists for the characters permitted in data items of usage display; this correspondence is used in converting alphanumeric characters to national characters in locale-based evaluation of a relation condition.

The locale categories LC_MESSAGES and LC_NUMERIC are not used directly by COBOL; however, the ability to set and query these locale categories is provided so that applications may use it.

The set of cultural elements constituting LC_ALL may include categories and cultural elements not used by COBOL.

8.2.1 Locale field names

Several locale field names are referenced in the COBOL specification in order to clarify processing. These and other relevant locale fields are defined in ISO/IEC 9945. The following locale field names are explicitly referenced:

Category	Field name	Description
LC_MONETARY		
	int_curr_symbol	international currency symbol
	currency_symbol	local currency symbol
	mon_decimal_point	decimal delimiter
	mon_thousands_sep	string used to group digits to the left of the decimal delimiter
	mon_grouping	size of each group of digits
	positive_sign	string used to indicate nonnegative valued quantities
	negative_sign	string used to indicate negative-valued quantities
	int_frac_digits	number of fractional digits to the right of the decimal delimiter
	frac_digits	number of fractional digits to the right of the decimal delimiter
	p_cs_precedes	indicator of whether the currency symbol precedes or succeeds the value for a nonnegative quantity
	n_cs_precedes	indicator of whether the currency symbol precedes or succeeds the value for a negative quantity
LC_TIME		
	d_fmt	date representation
	t_fmt	time representation

8.3 Lexical elements

The lexical elements are character-strings and separators.

8.3.1 Character-strings

A character-string is a character or a sequence of contiguous characters that forms a COBOL word, a literal, or a picture character-string. A character-string is delimited by separators.

8.3.1.1 COBOL words

A COBOL word is a character-string of not more than 31 characters that forms a compiler-directive word, a context-sensitive word, an intrinsic-function-name, a reserved word, a system-name, or a user-defined word. Each character of a COBOL word that is not a special character word shall be selected from the set of basic letters, basic digits, extended letters, and the basic special characters hyphen and underscore. The hyphen or underscore shall not appear as the first or last character in such words.

Within a compilation group, compilation-variable-names form intersecting sets with other types of user-defined words, system-names, context-sensitive words, and intrinsic-function names. The same COBOL word may be used as a compilation-variable-name and as one of these other types of words.

Within a source element the following apply:

1) Reserved words shall not be used as user-defined words or system-names.

2) Compiler-directive words, except those that are also reserved words, may be used as user-defined words and system-names.

3) Context-sensitive words may be used as user-defined words and system-names in contexts other than the language construct in which they are defined. Specific rules may apply to the interpretation of a word as user-defined or context-sensitive.

4) A given word may be used as a system-name and as a user-defined word, subject to the rules specified in 8.3.1.1.1, User-defined words, and 8.3.1.1.2, System-names.

5) Intrinsic-function-names may be used as user-defined words and system-names, except for

   — intrinsic function names LENGTH, RANDOM, SIGN, and SUM; and
   — intrinsic function names identified in a function-specifier in the REPOSITORY paragraph.

8.3.1.1.1 User-defined words

A user-defined word is a COBOL word that is supplied by the user to satisfy the format of a clause or statement.

The types of user-defined words are:

- alphabet-name
- any-length-structure-name
- class-name (for object orientation)
- class-name (for truth value proposition)
- compilation-variable-name
- condition-name
- constant-name
- data-name
- file-name
- function-prototype-name
- index-name
- interface-name

- level-number
- locale-name
- method-name
- mnemonic-name
- ordering-name
- paragraph-name
- parameter-name
- program-name
- program-prototype-name
- property-name
- record-key-name
- record-name
- report-name
- screen-name
- section-name
- symbolic-character
- type-name
- user-function-name

Within a source element, a given user-defined word may be used as only one type of user-defined word with the following exceptions:

1) a compilation-variable-name may be the same as any other type of user-defined word

2) a level-number may be the same as a paragraph-name or a section-name

3) the same name may be used as any of the following types of user-defined words:

   - constant-name
   - data-name
   - property-name
   - record-key-name
   - record-name

Further rules for uniqueness are specified in 8.4.1, Uniqueness of reference.

With the exception of section-names, paragraph-names, and level-numbers, each user-defined word shall contain at least one basic letter or extended letter. Level-numbers need not be unique; a given specification of a level-number may be identical to any other level-number.

The following user-defined words shall be externalized to the operating environment:

1) program-names of outermost programs, class-names, function-prototype-names, interface-names, method-names, program-prototype-names, property-names, and user-function-names

2) data-names, file-names, and record-names of items described with the EXTERNAL attribute.

The implementor shall specify whether extended letters may be specified in user-defined words externalized to the operating environment.

For any externalized user-defined words for which the AS phrase is specified, the content of the literal specified in that AS phrase is a name that is externalized to the operating environment. The implementor defines the formation and mapping rules of these names.

NOTE The AS phrase provides a way to specify names that are either case-sensitive or not valid COBOL words. Such names may be required by other programming languages or system components.

For any externalized user-defined words for which the AS phrase is not specified, the implementor defines the mapping between the user-defined word and the corresponding name that is externalized to the operating environment.

Within a run unit, all instances of a given name that is externalized to the operating environment shall identify the same kind of entity or item. Except for method-names and property-names, when two or more source elements identify something with the same externalized name, they refer to the same instance.

Externalized names shall be referenced in a source element only:

1) in the AS phrase in a repository paragraph entry,

2) in the AS phrase in an EXTERNAL clause,

3) as program-name in a CALL statement,

4) as program-name in a CANCEL statement,

5) as program-name in a program-address-identifier,

6) as method-name in an INVOKE statement or inline method invocation.

All other references to names for which externalization is permitted shall be specified using the user-defined words, as opposed to the externalized names.

In the AS phrases, only the externalized names shall be referenced. In the CALL, CANCEL, and INVOKE statements, the inline method invocation, and in the program-address-identifier, either the externalized names or the user-defined words may be referenced, depending on the conditions described below.

When an INVOKE statement or an inline method invocation references a method-name using a universal object reference:

1) When the COBOL call convention is implied or COBOL is specified in a CALL-CONVENTION compiler directive, that method-name is treated as a COBOL word that maps to the externalized name of the method to be invoked, applying the same implementor-defined mapping rules as for a method-name for which no AS phrase is specified.

2) When call-convention-name-1 is specified in an applicable CALL-CONVENTION compiler directive, that method-name is treated as a literal that maps, in a manner defined by the implementor, to the externalized name of the method to be invoked.

When an INVOKE statement or an inline method invocation references a method-name using an object reference that is not a universal object reference, the naming convention and mapping to be used for the method-name is determined by the entry convention of the class or interface that contains the method.

> NOTE If the method being invoked is referenced using a universal object reference, the ENTRY-CONVENTION, if any, in the class or interface definition containing the method is ignored.

When a CALL statement, a CANCEL statement, or a program-address-identifier references a program-name that names a compilation unit:

1) If the CALL statement, CANCEL statement, or program-address-identifier specifies a program-prototype-name, the naming convention and mapping used for the program-name is determined by the entry convention indicated by the description of the program to be called, as specified in 12.3.7, REPOSITORY paragraph, general rule 10;

2) otherwise:

a) When the COBOL call convention is implied or COBOL is specified in the CALL-CONVENTION compiler directive, the program-name is treated as a COBOL word that maps to the externalized name of the program, applying the same implementor-defined mapping rules as for a program-name for which no AS phrase is specified.

b) When call-convention-name-1 is specified in the CALL-CONVENTION compiler directive, the program-name is treated as a literal that maps, in a manner defined by the implementor, to the externalized name of the program.

8.3.1.1.1.1 Alphabet-name

An alphabet-name identifies a specific character set or collating sequence, or both. This relationship is established in the SPECIAL-NAMES paragraph.

8.3.1.1.1.2 Any-length-structure-name

An any-length-structure-name specifies the physical layout of an any-length elementary item. This relationship is established in the SPECIAL-NAMES paragraph.

8.3.1.1.1.3 Class-name (for object orientation)

A class-name identifies a class, the entity that defines common behavior and implementation for zero, one, or more objects.

8.3.1.1.1.4 Class-name (for truth value proposition)

A class-name identifies a proposition, for which a truth value can be determined, that the content of a data item consists exclusively of those characters listed in the definition of the class-name.

8.3.1.1.1.5 Compilation-variable-name

A compilation-variable-name identifies a compilation variable defined in a DEFINE compiler directive.

8.3.1.1.1.6 Condition-name

A condition-name identifies a value, set of values, or range of values defined in the data division, or identifies an on or off status defined in the SPECIAL-NAMES paragraph.

8.3.1.1.1.7 Constant-name

A constant-name identifies a constant, which is defined by a constant entry in the data division.

8.3.1.1.1.8 Data-name

A data-name identifies a data item described in a data description entry or a record described in a record description entry.

8.3.1.1.1.9 File-name

A file-name identifies a file connector described in a file description entry or a sort-merge file description entry within the file section of the data division.

8.3.1.1.1.10 Function-prototype-name

A function-prototype-name identifies a function prototype.

8.3.1.1.1.11 Index-name

An index-name identifies an index associated with a specific table.

8.3.1.1.1.12 Interface-name

An interface-name identifies an interface, a grouping of method prototypes.

8.3.1.1.1.13 Level-number

A level-number, expressed as a one-digit or two-digit number, indicates the hierarchical position of a data item or the special properties of a data description entry.

8.3.1.1.1.14 Locale-name

A locale-name identifies a locale that specifies a set of cultural elements. A locale-name is defined in the SPECIAL-NAMES paragraph.

8.3.1.1.1.15 Method-name

A method-name identifies a method.

8.3.1.1.1.16 Mnemonic-name

A mnemonic-name identifies an implementor-defined device-name, feature-name, or switch-name. This relationship is established in the SPECIAL-NAMES paragraph.

8.3.1.1.1.17 Ordering-name

An ordering-name identifies an ordering table in compliance with ISO/IEC 14651, used for the execution of the STANDARD-COMPARE intrinsic function.

8.3.1.1.1.18 Paragraph-name

A paragraph-name identifies a paragraph in the procedure division. Paragraph-names are equivalent if they are composed of the same sequence of the same number of COBOL characters.

NOTE The paragraph-names '00123' and '123' are different paragraph-names.

8.3.1.1.1.19 Parameter-name

A parameter-name identifies a formal parameter of a parameterized class or a parameterized interface.

8.3.1.1.1.20 Program-name

A program-name identifies a program. For a COBOL program, program-name is the name specified in the PROGRAM-ID paragraph of the program's identification division. For a non-COBOL program, the rules for formation of the program-name are defined by the implementor.

8.3.1.1.1.21 Program-prototype-name

A program-prototype-name identifies a program prototype.

8.3.1.1.1.22 Property-name

A property-name identifies a means of getting information out of and passing information back into an object.

8.3.1.1.1.23 Record-key-name

A record-key-name identifies a key associated with an indexed file.

8.3.1.1.1.24 Record-name

A record-name identifies a record described in a record description entry. A record-name may be specified where a data-name is allowed unless specific rules for the format disallow it.

8.3.1.1.1.25 Report-name

A report-name identifies a report described in a report description entry within the report section of the data division.

8.3.1.1.1.26 Screen-name

A screen-name identifies a screen description entry in the screen section.

8.3.1.1.1.27 Section-name

A section-name identifies a section in the procedure division.

8.3.1.1.1.28 Symbolic-character

A symbolic-character is a user-defined figurative constant that represents a value specified in the SPECIAL-NAMES paragraph.

8.3.1.1.1.29 Type-name

A type-name identifies a type declaration specified by a data description entry.

8.3.1.1.1.30 User-function-name

A user-function-name identifies a function.

8.3.1.1.2 System-names

A system-name is used to communicate with the operating environment. The implementor may define rules for the formation of a system-name that add restrictions to the rules for formation of a COBOL word.

The types of system-names are:

- call-convention-name
- code-name
- computer-name
- device-name
- entry-convention-name
- external-locale-name
- feature-name
- library-name
- physical-structure-name
- switch-name
- text-name

Within an implementation, a given system-name shall not belong to more than one of the following types of system-names: device-name, feature-name, and switch-name.

8.3.1.1.2.1 Call-convention-name

A call-convention-name identifies an implementor-defined convention for mapping a method-name or a program-name to its externalized name and may identify attributes of the linkage mechanism used to interact with a function, method, or program.

8.3.1.1.2.2 Code-name

A code-name identifies a character code set and a collating sequence.

8.3.1.1.2.3 Computer-name

A computer-name may identify the computer upon which the compilation unit is to be compiled or the runtime module is to be run.

8.3.1.1.2.4 Device-name

A device-name identifies a hardware or software device in the operating environment.

8.3.1.1.2.5 Entry-convention-name

An entry-convention-name identifies attributes of the linkage mechanism by which a function, method, or program is to receive control.

8.3.1.1.2.6 External-locale-name

An external-locale-name identifies a locale that specifies a set of cultural elements. This locale is provided in the operating environment.

8.3.1.1.2.7 Feature-name

A feature-name identifies a feature of an input-output device.

8.3.1.1.2.8 Library-name

A library-name identifies a COPY library.

8.3.1.1.2.9 Physical-structure-name

A physical-structure-name identifies an implementor-defined physical layout of an any-length elementary item.

8.3.1.1.2.10 Switch-name

A switch-name identifies an implementor-defined external switch.

8.3.1.1.2.11 Text-name

A text-name identifies a library text.

8.3.1.1.3 Reserved words

The COBOL words shown in 8.9, Reserved words, are reserved for use as keywords, optional words, or special-character words in language constructs. Reserved words shall not be used as system-names or user-defined words.

In order to reduce conflict between reserved words and user-defined words, the following rules apply to the formation of reserved words in this committee draft International Standard.

NOTE It is intended to apply these rules in future editions. It is recommended that implementors follow these rules in defining extensions.

1) Reserved words shall not begin with the digits 0 through 9 or the letters 'X', 'Y', or 'Z', except for the words ZERO, ZEROES, and ZEROS.

2) Reserved words shall be composed of at least two basic letters, except for special-character words.

3) Reserved words shall not begin with 1 or 2 letters followed by a hyphen, except for the words I-O and I-O-CONTROL and words beginning with 'B-'.

4) Reserved words shall not be formed with two or more consecutive hyphens.

The types of reserved words are:

— required words
— optional words

8.3.1.1.3.1 Required words

A required word is a word whose presence is required when the format in which the word appears is used.

Required words are of two types:

1) Keywords. Within each format, such words are uppercase and underlined.

2) Special character words. The special character words are:

Word	Meaning
+	Arithmetic operator - unary plus or addition
–	Arithmetic operator - unary minus or subtraction
*	Arithmetic operator - multiplication
/	Arithmetic operator - division
**	Arithmetic operator - exponentiation
&	Concatenation operator
>	Relational operator - greater than
<	Relational operator - less than
=	Relational operator - equal and assignment operator in COMPUTE
==	Pseudo-text delimiter in COPY and REPLACE statements
>=	Relational operator - greater than or equal
<=	Relational operator - less than or equal
<>	Relational operator - not equal
*>	Comment indicator
>>	Compiler directive indicator
::	Method invocation operator

8.3.1.1.3.2 Optional words

Within each format, uppercase words that are not underlined are called optional words and may be specified at the user's option with no effect on the semantics of the format.

8.3.1.1.4 Context-sensitive words

A context-sensitive word is a COBOL word that is reserved only in the general formats in which it is specified. Context-sensitive words and the contexts in which they are reserved are specified in 8.10, Context-sensitive words.

8.3.1.1.5 Intrinsic-function-names

An intrinsic-function-name is a COBOL word that identifies a specific intrinsic function. The list of intrinsic function names is given in 8.11, Intrinsic function names.

8.3.1.1.6 Exception-names

An exception-name is a COBOL word that identifies an exception condition. The list of exception-names is given in 14.6.12.1, Exception conditions.

8.3.1.2 Literals

A literal is defined by a reserved word that references a figurative constant or is a character-string representing a data value derived from the ordered set of characters of which the literal is composed. Each literal possesses a class and category: alphanumeric, boolean, national, or numeric.

The paired quotation symbols specified in the opening and closing delimiters of alphanumeric, boolean, and national literals may be either apostrophes or quotation marks. Both forms may be used within a single source unit. If the opening and closing delimiters are contiguous, the length of the literal is zero, and it is known as a zero-length literal.

Hexadecimal digits are used to specify the value of the literal in the hexadecimal-alphanumeric, hexadecimal-boolean, and hexadecimal-national formats of literals. The hexadecimal digits are the basic digits '0' through '9' and the basic letters 'A' through 'F'.

8.3.1.2.1 Alphanumeric literals

Alphanumeric literals are of the class and category alphanumeric.

8.3.1.2.1.1 General format

Format 1 (alphanumeric):

```
{ " [character-1]  ... " }
{ ' [character-1]  ... ' }
```

Format 2 (hexadecimal-alphanumeric):

```
{ X"[hex-character-sequence-1]  ..." }
{ X'[hex-character-sequence-1]  ...' }
```

8.3.1.2.1.2 Syntax rules

ALL FORMATS

1) The length of an alphanumeric literal, excluding the separators that delimit the literal, shall be less than or equal to 8,191 alphanumeric character positions.

FORMAT 1

2) Character-1 may be any character in the coded character set that the implementor has chosen for source code representation and designated as a character in the compile-time alphanumeric coded character set.

NOTE This allows, but does not require, characters in an alphanumeric literal to be represented in source code in a national coded character set. This permits, for example, a literal of the form "ABC" to be represented in the source code

in UTF-16 and stored as ISO/IEC 646. This is essential in order to allow source code to be represented entirely in a coded character set such as UTF-16, but is not restricted to that case.

An implementation is neither required to recognize nor prohibited from recognizing UTF-8 or mixed alphanumeric and national characters in format 1 alphanumeric literals. When permitted, the capability shall be optionally available to the user in a manner that does not restrict the characters normally recognized by that implementation; the implementor shall specify any applicable syntax rules.

3) Two contiguous quotation symbol characters matching the quotation symbol used in the opening delimiter represent a single occurrence of that quotation symbol character in the content of the literal.

4) The two contiguous quotation symbols used to represent a single quotation symbol character shall be in the same coded character set representation as the opening quotation symbol.

FORMAT 2

5) Hex-character-sequence-1 shall be composed of hexadecimal digits.

NOTE Hexadecimal-alphanumeric literals may be of zero length.

6) Each hex-character-sequence-1 shall consist of the number of hexadecimal digits that the implementor has specified as the number of hexadecimal digits that map to an alphanumeric character.

8.3.1.2.1.3 General rules

ALL FORMATS

1) The separators that delimit the alphanumeric literal are not included in the value of the alphanumeric literal.

2) Alphanumeric literals are of the class and category alphanumeric.

FORMAT 1

3) If character-1 is not specified, the literal is a zero-length literal. If character-1 is specified, the following rules apply:

   a) The value of the literal at compile time is the string of occurrences of character-1, represented in the computer's compile-time coded character set defined by the implementor for usage DISPLAY.

   b) The value of the literal at runtime is the string of alphanumeric characters that results from converting the compile-time value of the literal to its runtime equivalent, as described in 8.1.1, Computer's coded character set.

      NOTE This rule permits storing alphanumeric literals in national character representation when usage DISPLAY is implemented in a large character set such as the UCS.

   c) When the implementor provides the option of UTF-8 or mixed alphanumeric and national characters in the content of format 1 alphanumeric literals, the implementor shall specify the applicable general rules.

FORMAT 2

4) If hexadecimal-sequence-1 is not specified the literal is a zero-length literal, otherwise the value of the literal at runtime shall be a string of alphanumeric characters, each of which has the bit configuration specified by one occurrence of hex-character-sequence-1.

   The implementor defines the result of specifying a hex-character-sequence-1 for which no corresponding character in that coded character set exists. The implementor also defines the mapping of each hex-character-sequence-1 to a character, when the characters do not occupy a multiple of four bits.

8.3.1.2.2 Numeric literals

Numeric literals are of the class and category numeric.

8.3.1.2.2.1 Fixed-point numeric literals

A fixed-point numeric literal is a character-string whose characters are selected from the digits '0' through '9', the plus sign, the minus sign, and the decimal point. The implementor shall allow for fixed-point numeric literals of 1 through 31 digits in length. The rules for the formation and value of fixed-point numeric literals are as follows:

1) A literal shall contain at least one digit.

2) A literal shall not contain more than one sign character. If a sign is used, it shall appear as the leftmost character of the literal. If the literal is unsigned, the literal is nonnegative.

3) A literal shall not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character.

4) The value of a fixed-point numeric literal is the algebraic quantity represented by the characters in the fixed-point numeric literal. The size of a fixed-point numeric literal is equal to the number of digits in the string of characters in the literal.

An integer literal is a fixed-point numeric literal that contains no decimal point.

8.3.1.2.2.2 Floating-point numeric literals

The rules for the formation and value of floating-point numeric literals are:

1) A floating-point numeric literal is formed from two fixed-point numeric literals separated by the letter 'E' without intervening spaces.

2) The literal to the left of the 'E' represents the significand. It may be signed and shall include a decimal point. The significand shall be from 1 to 34 digits in length. If the significand is signed, the floating-point numeric literal is considered to be signed. If the significand is unsigned, the floating-point numeric literal is considered to be positive.

3) The literal to the right of the 'E' represents the exponent. It may be signed and shall have a maximum of four digits and no decimal point. The maximum permitted value and minimum permitted value of the exponent is implementor-defined.

4) If all the digits in the significand are zero, then all the digits of the exponent shall also be zero and neither significand nor exponent shall have a negative sign.

5) The value of a floating-point numeric literal is the algebraic product of the value of its significand and the quantity derived by raising ten to the power of the exponent.

8.3.1.2.3 Boolean literals

Boolean literals are of the class and category boolean.

8.3.1.2.3.1 General format

Format 1 (boolean)

```
{ B"[boolean-character-1] ... " }
{ B'[boolean-character-1] ... ' }
```

Format 2 (hexadecimal-boolean)

```
{ BX"[hexadecimal-digit-1] ... " }
{ BX'[hexadecimal-digit-1] ... ' }
```

8.3.1.2.3.2 Syntax rules

ALL FORMATS

1) The length of a boolean literal, excluding the separators that delimit the literal, shall be less than or equal to 8,191 boolean character positions.

FORMAT 1

2) Boolean-character-1 shall be a boolean character, '0' or '1', from the computer's coded character set.

FORMAT 2

3) Hexadecimal-digit-1 shall be a hexadecimal digit.

NOTE Hexadecimal-boolean literals may be of zero length.

8.3.1.2.3.3 General rules

ALL FORMATS

1) The separators that delimit the boolean literal are not included in the value of the boolean literal.

2) Boolean literals are of the class and category boolean.

FORMAT 1

3) If Boolean-character-1 is specified, the value of a boolean literal is the value of the sequence of occurrences of boolean-character-1.

FORMAT 2

4) Each hexadecimal digit has the following boolean equivalent value: '0' is B"0000", '1' is B"0001", '2' is B"0010", '3' is B"0011", '4' is B"0100", '5' is B"0101", '6' is B"0110", '7' is B"0111", '8' is B"1000", '9' is B"1001", 'A' is B"1010", 'B' is B"1011", 'C' is B"1100", 'D' is B"1101", 'E' is B"1110", 'F' is B"1111".

5) The value of the literal at runtime is the value of an equivalent boolean literal formed by replacing each hexadecimal digit by its boolean equivalent value, and replacing the leading BX" separator with the B" separator.

8.3.1.2.4 National literals

National literals are of the class and category national.

8.3.1.2.4.1 General format

Format 1 (national)

```
{ N"[character-1] ... " }
{ N'[character-1] ... ' }
```

Format 2 (hexadecimal-national)

```
{ NX"[hex-character-sequence-1]  ... "  }
{ NX'[hex-character-sequence-1]  ... '  }
```

8.3.1.2.4.2 Syntax rules

ALL FORMATS

1) The length of a national literal, excluding the separators that delimit the literal, shall be less than or equal to 8,191 national character positions.

2) Character-1 may be:

   — any character in the national coded character set that the implementor has designated for source code representation

   — any character in the alphanumeric coded character set that the implementor has designated for source code representation such that a correspondence exists between that alphanumeric character and a national character.

   NOTE The implementor may choose to represent the source code entirely in a national coded character set or in a mix of alphanumeric and national coded character sets. The content of a national literal can be coded in either representation and stored in national representation. For example, a literal of the form N'ABC' can be represented in single-byte characters and stored as UTF-16. This is essential in order to allow source code to be represented in a coded character set such as UTF-8, but is not restricted to that case.

FORMAT 1

3) Two contiguous quotation symbol characters matching the quotation symbol used in the opening delimiter represent a single occurrence of that quotation symbol character in the content of the literal. The two contiguous quotation symbol characters shall be in the same coded character set representation as the opening quotation symbol.

FORMAT 2

4) Hex-character-sequence-1 shall be composed of hexadecimal digits.

5) Each hex-character-sequence-1 shall consist of the number of hexadecimal digits that the implementor has specified as the number of hexadecimal digits that map to a national character.

NOTE Hexadecimal-national literals may be of zero length.

8.3.1.2.4.3 General rules

ALL FORMATS

1) The separators that delimit the national literal are not included in the value of the national literal.

2) National literals are of the class and category national.

FORMAT 1

3) If character-1 is not specified, the literal is a zero-length literal. If character-1 is specified, the following rules apply:

a) The value of the literal at compile time is the string of occurrences of character-1, represented in the computer's compile-time coded character set defined by the implementor for usage NATIONAL. Control functions, if any, that switch character set encoding are not included in the value of the literal.

b) The value of the literal at runtime is the string of national characters that results from converting the compile-time value of the literal to its runtime equivalent, as described in 8.1.1, Computer's coded character set.

FORMAT 2

4) If hex-character-sequence-1 is not specified the literal is a zero-length literal, else the value of the literal at runtime shall be a string of national characters, each of which has the bit configuration specified by one occurrence of hex-character-sequence-1.

The implementor defines the result of specifying a hex-character-sequence-1 for which no corresponding character in that coded character set exists. The implementor also defines the mapping of each hex-character-sequence-1 to a character, when the characters do not occupy a multiple of four bits.

8.3.1.2.5 Figurative constant values

Figurative constant values are generated by the compiler and referenced through the use of the reserved words given below.

8.3.1.2.5.1 General format

Format 1 (zero):

ALL { ZERO / ZEROES / ZEROS }

Format 2 (space):

ALL { SPACE / SPACES }

Format 3 (high-value):

ALL { HIGH-VALUE / HIGH-VALUES }

Format 4 (low-value):

ALL { LOW-VALUE / LOW-VALUES }

Format 5 (quote):

ALL { QUOTE / QUOTES }

Format 6 (all-literal):

ALL literal-1

Format 7 (symbolic-character):

ALL symbolic-character-1

8.3.1.2.5.2 Syntax rules

ALL FORMATS

1) A figurative constant may be used whenever 'literal' appears in a format or when a rule allows it, with the following restrictions:

   a) If the literal is restricted to a numeric literal, the only figurative constant permitted is ZERO (ZEROS, ZEROES) without the ALL phrase.

   b) A figurative constant shall not be specified where a syntax rule prohibits it.

FORMAT 6

2) Literal-1 shall be an alphanumeric, boolean, or national literal, any of which may be a concatenation expression. The literal shall be neither a figurative constant nor a zero-length literal.

3) If the length of literal-1 is greater than one, it is not permitted to be associated with a numeric or numeric-edited item.

FORMAT 7

4) Symbolic-character-1 shall be specified in the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph.

8.3.1.2.5.3 General rules

ALL FORMATS

1) When a figurative constant is used in a context requiring national characters, the figurative constant represents a national character value. Otherwise, when a figurative constant represents a character value, the figurative constant represents an alphanumeric character value. In both cases, the character value representation of the figurative constant ZERO (ZEROS, ZEROES), SPACE (SPACES), and QUOTE (QUOTES) is the value of the character '0', space, and '"', respectively, in the computer's runtime coded character set. The implementor shall specify the unique representation of ZERO, SPACE, and QUOTE in the computer's alphanumeric and national coded character sets.

2) When a figurative constant represents a string of one or more characters and the length of the string is not specified in the rules for the context in which the figurative constant is used, the length of the string is determined from context by applying the following rules in order:

   a) When a figurative constant is specified in a concatenation expression, the length of the string is one character.

   b) When a figurative constant is specified in a VALUE clause or in association with a data item, literal, or intermediate result, the string of characters is repeated character by character until the size of the resultant string is greater than or equal to the number of character positions in the associated data item, literal, or intermediate result. This resultant string is then truncated from the right until the number of character positions remaining is equal either to 1 or to the number of character positions in the associated data item,

literal, or intermediate result, whichever is greater. This is done prior to and independent of the application of any JUSTIFIED clause that may be associated with the data item.

NOTE A figurative constant is associated with a data item or literal when, for example, the figurative constant is moved to it, compared with it, or paired with it in a binary operation.

c) When a figurative constant is other than ALL literal-1, the length of the string is one character.

NOTE For example, when the figurative constant appears in a DISPLAY, STOP, STRING, or UNSTRING statement, it is one character.

d) The length of the string is the length of literal-1.

FORMAT 1

3) The zero format represents the numeric value '0', one or more of the boolean character '0', or one or more of the character '0' in the computer's runtime coded character set, depending on context.

FORMAT 2

4) The space format represents one or more of the character space in the computer's runtime coded character set.

FORMAT 3

5) At compile time and when referenced in the SPECIAL-NAMES paragraph, the high-value format represents the character, or multiple-character combination, that has the highest ordinal position in the collating sequence used during compilation.

At runtime, when referenced outside the SPECIAL-NAMES paragraph, the high-value format represents the character, or multiple-character combination, that has the highest ordinal position in the runtime collating sequence.

When locale category LC_COLLATE is in effect for the program collating sequence, HIGH-VALUES is the character, or multiple-character combination, that has the highest ordinal position in the collating sequence specified by the locale in effect.

If the context of the figurative constant requires national characters, the national program collating sequence is used; otherwise, the alphanumeric program collating sequence is used.

FORMAT 4

6) At compile time and when referenced in the SPECIAL-NAMES paragraph, the low-value format represents the character, or multiple-character combination, that has the lowest ordinal position in the collating sequence used during compilation.

At runtime, when referenced outside the SPECIAL-NAMES paragraph, the low-value format represents the character, or multiple-character combination, that has the lowest ordinal position in the runtime collating sequence.

When locale category LC_COLLATE is in effect for the program collating sequence, LOW-VALUES is the character, or multiple-character combination, that has the lowest ordinal position in the collating sequence specified by the locale in effect.

If the context of the figurative constant requires national characters, the national program collating sequence is used; otherwise, the alphanumeric program collating sequence is used.

FORMAT 5

7) The quote format represents one or more of the quotation mark character (' " ') in the computer's runtime coded character set. The word QUOTE or QUOTES shall not be used in place of a quotation symbol to bound a literal.

FORMAT 6

8) The all-literal format represents all or part of the string generated by successive concatenations of the characters comprising literal-1.

FORMAT 7

9) The symbolic-character format represents one or more of the character specified as the value of symbolic-character-1 in the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph.

8.3.1.3 Picture character-strings

A picture character-string consists of certain symbols that are composed of the currency symbol and certain combinations of characters in the COBOL character repertoire. An explanation of the picture character-string and the rules that govern its use are given in 13.18.39, PICTURE clause.

8.3.2 Separators

A separator is one of the following, except when appearing in a literal:

1) The COBOL character space is a separator. Anywhere a space is used as a separator or as part of a separator, more than one space may be used. All spaces immediately following the separators comma, semicolon, or period are considered part of that separator and are not considered to be the separator space.

2) The COBOL characters comma and semicolon, immediately followed by a space, are separators that may be used anywhere the separator space is used. They can be used to improve readability.

3) The COBOL character period, when followed by a space, is a separator. The separator period shall be used only to indicate the end of a sentence, or as shown in formats.

4) Except when appearing in a picture character -string, the COBOL characters right parenthesis and left parenthesis are separators. Except in pseudo-text, parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, a list of function or method arguments, a reference modifier, arithmetic or boolean expressions, or conditions.

5) The opening delimiters and closing delimiters of literals are separators. Either an apostrophe or a quotation mark may be used as the quotation symbol character in opening and closing delimiters.

   The opening delimiters of literals are:

   — a quotation symbol
   — the two contiguous characters B", B', N", N', X", and X'
   — the three contiguous characters BX", BX', NX", and NX'

   The closing delimiters of literals are:

   — a quotation mark when the opening delimiter uses a quotation mark
   — an apostrophe when the opening delimiter uses an apostrophe

   The opening delimiter shall be immediately preceded by a space, left parenthesis, or opening pseudo-text delimiter. The closing delimiter shall be immediately followed by one of the separators space, comma, semicolon, period, right parenthesis, or closing pseudo-text delimiter. Separators immediately preceding the opening delimiter are not part of the opening delimiter. Separators immediately following the closing delimiter are not part of the closing delimiter.

6) Pseudo-text delimiters are separators. An opening pseudo-text delimiter shall be immediately preceded by a space; a closing pseudo-text delimiter shall be immediately followed by one of the separators space, comma, semicolon, or period. Pseudo-text delimiters may appear only in balanced pairs delimiting pseudo-text.

7) The COBOL character colon, except as part of the invocation operator, is a separator and is required when shown in the general formats.

8) The separator space may optionally immediately precede all separators except:

   a) As specified by reference format rules (see 6, Reference format.)

   b) The closing delimiter of a literal. In this case, a preceding space is considered as part of the literal and not as a separator.

   c) The opening pseudo-text delimiter, where the preceding space is required.

   d) The terminating period of a data description entry that ends with a PICTURE clause that contains a comma or period as the last character of the picture character-string.

9) The separator space may optionally immediately follow any separator except the opening delimiter of a literal. A space following the opening delimiter of a literal is part of the literal and not a separator.

8.4 References

References identify elements referred to during compilation of source unit or execution of a run unit. The reserved words and types of names specified in 8.3, Lexical elements, are forms of reference. Additional forms of reference are identifiers and condition-names.

8.4.1 Uniqueness of reference

Every user-defined name in a source element is assigned, by the user, to name a resource that is to be used in solving a data processing problem. (See 8.3.1.1.1, User-defined words.) In order to use a resource, a statement shall contain a reference that uniquely identifies that resource. In order to ensure uniqueness of reference, a user-defined name may be qualified, subscripted, or reference modified as described in the following paragraphs.

When the same name has been assigned in separate source elements to two or more occurrences of a resource of a given type, and when qualification by itself does not allow the reference in one of those source elements to differentiate between the two identically named resources, then certain conventions that limit the scope of names apply. These conventions ensure that the resource identified is that described in the source element containing the reference. (See 8.4.5, Scope of names.)

The same name shall not be used both as the name of an external record and as the name of any other external data item described in the run unit. The same name shall not be used both as the name of an item possessing the global attribute and as the name of any other data item described in the source element that describes that global data item.

8.4.1.1 Qualification

Qualification is used to allow unique reference of user names. Qualification is the specification of superordinate names from the hierarchy to which a user-defined name belongs. The superordinate names are called qualifiers. Identical user-defined names may be specified in a source unit; however, uniqueness shall be established through qualification for each user-defined name explicitly referenced, except as specified in rules 2 through 6. All available qualifiers need not be specified so long as uniqueness is established.

Qualification of a user-defined name is required unless one of the following is true:

1) No other name has the identical spelling.

2) It is unique within the context of a REDEFINES clause.

3) It is unique within the context of a VARYING clause.

4) Any other definition of the name is subordinate to a type declaration entry for which the type-name is not referenced in any TYPE clause in the source unit.

5) The name is a data-name referenced in a data description entry clause whose subject is subordinate to the same group item as that data-name. In this case, the names of any group items superordinate to both the data-name and the subject of the data description entry clause are used as implicit qualifiers for the reference, in addition to any explicit qualifiers needed to establish uniqueness within that group.

6) The name is a paragraph-name and the section containing the reference also contains the named paragraph.

8.4.1.1.1 General format

Format 1 (qualified-data-name):

data-name-1 [data-qualifier] ... [file-report-qualifier]

Format 2 (qualified-condition-name):

condition-name-1 [data-qualifier] ...[file-report-qualifier]

Format 3 (qualified-index-name):

index-name-1 [data-qualifier] ...[file-report-qualifier]

Format 4 (qualified-procedure-name):

paragraph-name-1 [{ IN / OF } section-name-1]

Format 5 (qualified-screen-name):

screen-name-1 [{ IN / OF } screen-name-2] ...

Format 6 (qualified-record-key-name):

record-key-name-1 [{ IN / OF } file-name-2]

Format 7 (qualified-linage-counter):

LINAGE-COUNTER [{ IN / OF } file-name-3]

Format 8 (qualified-report-counter):

{ PAGE-COUNTER / LINE-COUNTER } [{ IN / OF } report-name-2]

where data-qualifier is:

{ IN / OF } data-name-2

where file-report-qualifier is:

{ IN / OF } { file-name-1 / report-name-1 }

8.4.1.1.2 Syntax rules

1) For each nonunique user-defined name that is explicitly referenced, uniqueness shall be established through a sequence of qualifiers that precludes any ambiguity of reference.

2) A name may be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any such set may be used.

3) The words IN and OF are equivalent.

4) Each data-name-2 shall be the name associated with a level number to which the item being qualified is subordinate. Qualifiers shall be specified in the order of successively more inclusive levels in the hierarchy.

   NOTE For purposes of item qualification, the hierarchy of an item declared at level 88 includes the conditional variable with which it is associated, and progresses through the successively more inclusive levels of the hierarchy of that conditional variable.

5) The qualification of a condition-name may include the conditional variable with which the condition-name is associated, as well as by any name by which that conditional variable can be qualified.

6) The qualification of an index-name may include the name of the table with which the index-name is associated, as well as any name by which that table can be qualified.

7) If explicitly referenced, a paragraph-name shall not be duplicated within a section. A paragraph-name need not be qualified when referred to from within the same section.

8) LINAGE-COUNTER shall be qualified if more than one file description entry containing a LINAGE clause has been specified in the source element.

9) LINE-COUNTER shall be qualified each time it is referenced in the procedure division if more than one report description entry is specified in the source element. In the report section, an unqualified reference to LINE-COUNTER is qualified implicitly by the name of the report in whose report description entry the reference is made. Whenever the LINE-COUNTER of a different report is referenced, LINE-COUNTER shall be qualified explicitly by the report-name associated with the different report.

10) PAGE-COUNTER shall be qualified each time it is referenced in the procedure division if more than one report description entry is specified in the source element. In the report section, an unqualified reference to the PAGE-COUNTER is qualified implicitly by the name of the report in whose report description entry the reference is made. Whenever the PAGE-COUNTER of a different report is referenced, PAGE-COUNTER shall be qualified explicitly by the report-name associated with the different report.

8.4.1.2 Subscripts

Subscripts are used when reference is made to an individual element within a table of like elements.

8.4.1.2.1 General format

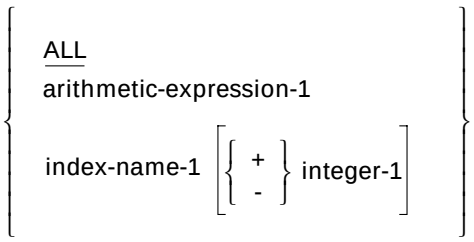
Format 1 (qualified-data-name-with-subscripts):

qualified-data-name-1 [(subscript ...)]

Format 2 (qualified-condition-name-with-subscripts):

qualified-condition-name-1 [(subscript ...)]

where subscript is:



NOTE Qualified-data-name-1 and qualified-condition-name-1 are shown for context and are not part of the subscript general format.

8.4.1.2.2 Syntax rules

1) Qualified-data-name-1 and qualified-condition-name-1 are defined in 8.4.1.1, Qualification.

2) If a subscript is specified, the data description entry describing qualified-data-name-1 or the conditional variable associated with qualified-condition-name-1 shall contain an OCCURS clause or shall be subordinate to a data description entry that contains an OCCURS clause.

3) Except as defined in syntax rule 5, when a reference is made to a table element, the number of subscripts shall equal the number of OCCURS clauses in the description of the table element being referenced. This allows a maximum of seven subscripts to be specified. When more than one subscript is required, the subscripts are written in the order of successively less inclusive dimensions of the table.

4) Index-name-1 shall correspond to a data description entry in the hierarchy of the table being referenced that contains an INDEXED BY phrase specifying that index-name.

5) Each table element reference shall be subscripted except when such reference appears:

   a) As the subject of a SEARCH statement.

   b) In a REDEFINES clause.

   c) In the KEY IS phrase of an OCCURS clause.

   d) In the KEY phrase of a SORT statement that references a table.

   e) As the subject of a SORT statement that references a table where the rightmost subscript is not the word ALL.

   f) In the FROM, TO, or USING clause of a screen description entry when the subject of the entry has an OCCURS clause.

   g) As a data-name addend in the SUM clause of a report description entry.

      NOTE When used as the subject of a SORT statement, the number of subscripts is equal to one less than the number of OCCURS clauses in the description of the table element being referenced, unless the rightmost subscript is the word ALL.

6) The subscript ALL may be used only:

   — When the subscripted identifier is used as an intrinsic function argument, or

— as the rightmost or only subscript of a table in the table format of a SORT statement. This is equivalent to omitting the rightmost or only subscript in this context.

7) ALL shall not be specified if qualified-condition-name-1 is specified.

8) In the report section, neither a sum counter nor the LINE-COUNTER and PAGE-COUNTER identifiers may be used as a subscript.

8.4.1.2.3 General rules

1) A subscript is determined as follows:

   a) If ALL is specified, the subscript is all of the possible values of a subscript for the associated table as specified in the rules for the functions for which the subscript ALL is allowed.

   b) If arithmetic-expression-1 is specified, the subscript is the result of the evaluation of arithmetic-expression-1. If the evaluation of arithmetic-expression-1 does not result in an integer, the EC-BOUND-SUBSCRIPT exception condition is set to exist.

   c) If index-name-1 is specified, the subscript is the occurrence number represented by the value of the index referenced by index-name-1 modified by integer-1, if specified. The mapping of the value of the index referenced by index-name-1 to an occurrence number is defined by the implementor. If integer-1 is specified, the subscript is the occurrence number derived from the index incremented by the value of integer-1 (when the operator + is used) or decremented by the value of integer-1 (when the operator – is used).

2) The value of a subscript shall be a positive integer. The lowest possible occurrence number represented by a subscript is 1, which identifies the first element of any given dimension of a table. Each successive element within that dimension of the table is referenced by occurrence numbers of 2, 3, ... The highest permissible occurrence number for any given dimension of a fixed-capacity or occurs-depending table is the maximum number of occurrences of the item as specified in the associated OCCURS clause. The highest permissible occurrence number for any given dimension of a dynamic-capacity table is implementor-defined and may be affected by the availability of runtime resources. If the value of the subscript is not a positive integer or is less than one or is greater than the highest permissible occurrence number, the EC-BOUND-SUBSCRIPT exception condition is set to exist.

8.4.2 Identifiers

8.4.2.1 Identifier

An identifier is a sequence of character-strings and separators used to reference a data item uniquely.

8.4.2.1.1 General format

Format 1 (function-identifier):

function-identifier-1

Format 2 (qualified-data-name-with-subscripts):

qualified-data-name-with-subscripts-1

Format 3 (reference-modification):

identifier-1 reference-modifier-1

Format 4 (inline-method-invocation):

inline-invocation-1

Format 5 (object-view):

identifier-2 object-view-1

Format 6 (predefined-object):

```
{ EXCEPTION-OBJECT          }
{ NULL                      }
{ SELF                      }
{ [ class-name-1 OF ] SUPER }
```

Format 7 (object-property):

property-name-1 OF identifier-3

Format 8 (predefined-address)

NULL

Format 9 (address-identifier)

```
{ data-address-identifier-1    }
{ program-address-identifier-1 }
```

Format 10 (qualified-linage-counter):

qualified-linage-counter-1

Format 11 (qualified-report-counter):

qualified-report-counter-1

8.4.2.1.2 Syntax rules

ALL FORMATS

1) Identifier is defined recursively: whenever the format for an identifier allows another identifier to be specified, that other identifier may be any of the formats for an identifier, including the one being defined provided the rules for each format are followed.

FORMAT 1

2) Function-identifier-1 is defined by 8.4.2.2, Function-identifier.

FORMAT 2

3) Qualified-data-name-with-subscripts-1 is defined by 8.4.1.2, Subscripts.

FORMAT 3

4) Reference-modifier-1 is defined by 8.4.2.3, Reference-modification.

FORMAT 4

5) Inline-invocation-1 is defined by 8.4.2.4, Inline method invocation.

FORMAT 5

6) Object-view-1 is defined by 8.4.2.5, Object-view.

FORMAT 6

7) Predefined-object references are defined by 8.4.2.6, EXCEPTION-OBJECT; 8.4.2.7, NULL object reference; and 8.4.2.8, SELF and SUPER.

FORMAT 7

8) Object properties are defined by 8.4.2.9, Object property.

FORMAT 8

9) Predefined-address NULL is defined in 8.4.2.10, NULL address pointer.

FORMAT 9

10) Address-identifiers are defined by 8.4.2.11, Data-address-identifier and 8.4.2.13, Program-address-identifier.

FORMAT 10

11) Qualified-linage-counter-1 is defined in 8.4.1.1, Qualification.

FORMAT 11

12) Qualified-report-counter-1 is defined in 8.4.1.1, Qualification.

8.4.2.1.3 General rules

1) The order in which the various components of an identifier are applied is as follows, with the first to be applied listed first:

   a) a qualified-data-name-with-subscript; a predefined-object reference; a function-identifier without arguments; a qualified-report-counter; or a qualified-linage-counter

   b) an address-identifier applies to an identifier on the right

   c) an object-view applies to the identifier on the left

   d) OF for object properties applies the property-name on the left to the identifier on the right

   e) the inline method invocation operator applies the literal method-name with optional arguments enclosed in parentheses on the right to the identifier on the left

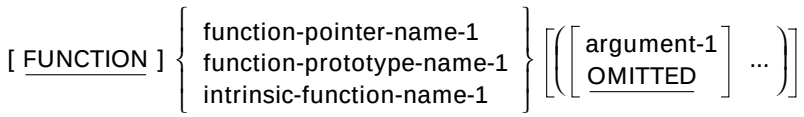
   f) a function-identifier with arguments applies the function-name on the left to a list of arguments enclosed in parentheses on the right

   g) a reference modifier applies to the identifier on the left.

8.4.2.2 Function-identifier

A function-identifier references the unique data item that results from the evaluation of a function.

8.4.2.2.1 General format



8.4.2.2.2 Syntax rules

1) A function-identifier shall not be specified as a receiving operand.

2) If intrinsic-function-name-1 or the ALL phrase is specified in the REPOSITORY paragraph or if function-prototype-name-1 or function-pointer-name-1 is specified, the word FUNCTION may be omitted from the function-identifier; otherwise the word FUNCTION is required.

3) Function-prototype-name-1 shall be the user-function-name of the containing function definition or a function prototype specified in the REPOSITORY paragraph.

4) Function-pointer-name-1 shall be defined as a function-pointer data item.

5) If function-pointer-name-1 is specified, the parentheses shall be specified.

6) If a function's definition permits arguments and a left parenthesis immediately follows function-prototype-name-1 or intrinsic-function-name-1, the left parenthesis is always treated as the left parenthesis of that function's arguments.

   NOTE For a function that may be referenced either with or without arguments, such as the RANDOM function, careful coding is necessary to ensure correct interpretation. For example, in the following:

   FUNCTION MAX (FUNCTION RANDOM (A) B)

   'A' is treated as an argument to the RANDOM function. If 'A' is instead meant to be a second argument to the MAX function, different coding is necessary - either:

   FUNCTION MAX ((FUNCTION RANDOM) (A) B)

   or

   FUNCTION MAX (FUNCTION RANDOM () A B)

   or

   FUNCTION MAX (FUNCTION RANDOM A B).

7) The word OMITTED shall not be specified if intrinsic-function-name-1 is specified.

8) Argument-1 shall be an identifier, a literal, a boolean expression, or an arithmetic expression. Specific rules governing the number, class, category, and type of argument-1 are given for intrinsic functions in the definition of that intrinsic function in 15, Intrinsic functions, and for user-defined functions in 14.8, Conformance for parameters and returning items.

9) If the word OMITTED is specified, the OPTIONAL phrase shall be specified for the corresponding formal parameter.

10) If function-prototype-name-1 or function-pointer-name-1 is specified and the formal parameter corresponding to argument-1 is specified with a BY VALUE phrase, argument-1 shall be of class numeric, object, or pointer.

11) A numeric function shall not be specified where an integer operand is required, even though a particular reference of the numeric function might yield an integer value.

12) An integer function other than the integer form of the ABS function shall not be specified where an unsigned integer is required.

13) If function-prototype-name-1 or function-pointer-name-1 is specified, the rules for conformance specified in 14.8, Conformance for parameters and returning items, apply.

14) If function-prototype-name-1 or function-pointer-name-1 is specified and the formal parameter corresponding to argument-1 is specified with the BY REFERENCE phrase in the USING phrase of the procedure division header and argument-1 is a bit data item, argument-1 shall be described such that:

    a) subscripting and reference modification in argument-1 consist of only fixed-point numeric literals or arithmetic expressions in which all operands are fixed-point numeric literals and the exponentiation operator is not specified; and

    b) it is aligned on a byte boundary.

8.4.2.2.3 General rules

1) A function-identifier references a temporary data item whose value is determined when the function is referenced at runtime.

   If intrinsic-function-name-1 is specified, the temporary data item is an elementary data item whose description and category are specified by the definition of that intrinsic function in 15, Intrinsic functions.

   If function-prototype-name-1 is specified, the description, class, and category of the temporary data item is that specified by the description in the linkage section of the item specified in the RETURNING phrase of the procedure division header of the function prototype identified by function-prototype-name-1.

   If function-pointer-name-1 is specified, the description, class, and category of the temporary data item is that specified by the description in the linkage section of the item specified in the RETURNING phrase of the procedure division header of the function prototype identified by the TO phrase of the USAGE clause in the definition of function-pointer-name-1.

2) At the time reference is made to a function, its arguments are evaluated individually in the order specified in the list of arguments, from left to right. An argument being evaluated may itself be a function-identifier or may be an expression containing function-identifiers. There is no restriction preventing the function referenced in evaluating an argument from being the same function as that for which the argument is specified. Additional rules for intrinsic functions are given in 15, Intrinsic functions, for user-defined functions in 14.2.3, General rules of the procedure division and in 14.8, Conformance for parameters and returning items.

3) If function-prototype-name-1 is specified, the function to be activated is identified by function-prototype-name-1 in accordance with the rules specified in 12.3.7, REPOSITORY paragraph, and function-prototype-name-1 is used to determine the characteristics of the activated function.

4) If function-pointer-name-1 is specified, the function prototype specified in the TO phrase of the USAGE clause in the definition of function-pointer-name-1 is used to determine the characteristics of the activated element and the function to be activated.

5) If function-prototype-name-1 or function-pointer-name-1 is specified, the manner used for passing each argument is determined as follows:

    a) BY REFERENCE is assumed when the BY REFERENCE phrase is specified or implied for the corresponding formal parameter and argument-1 is an identifier that is permitted as a receiving operand, other than an object property or object data item.

    b) BY CONTENT is assumed when the BY REFERENCE phrase is specified or implied for the corresponding formal parameter and argument-1 is a literal, an arithmetic expression, a boolean expression, an object property, object data item, or any identifier that is not permitted as a receiving operand.

c) BY VALUE is assumed when the BY VALUE phrase is specified for the corresponding formal parameter.

6) Evaluation of the function-identifier proceeds as follows:

a) Each argument-1 is evaluated at the beginning of the evaluation of the function-identifier. If an exception condition exists, no function is activated and execution proceeds as specified in general rule 6f. If an exception condition does not exist, the values of argument-1 are made available to the activated function at the time control is transferred to that function.

b) If function-prototype-name-1 or function-pointer-name-1 is specified, the runtime system attempts to locate the function being activated. If function-prototype-name-1 is specified, the rules are specified in 8.4.5, Scope of names and 8.4.5.5, Scope of function-prototype-names. Additional rules are given in 12.3.7, REPOSITORY paragraph. If the function is not found, the EC-PROGRAM-NOT-FOUND exception condition is set to exist, the function is not activated, and execution continues as specified in general rule 6f.

c) If the function is located but the resources necessary to execute the function are not available, the EC-PROGRAM-RESOURCES exception condition is set to exist, the function is not activated, and execution continues as specified in general rule 6f. The runtime resources that are checked in order to determine the availability of the function for execution are defined by the implementor. If function-pointer-name-1 is specified, the runtime system attempts to execute the function at the address pointed to by function-pointer-name-1. If function-pointer-name-1 is NULL, the EC-FUNCTION-PTR-NULL exception condition is set to exist, no function is activated, and execution continues as specified in general rule 6f.

d) The function specified by the function-identifier is made available for execution and control is transferred to the activated function in a manner consistent with the entry convention specified for the function. If function-prototype-name-1 or function-pointer-name-1 is specified and the function to be activated is a COBOL function, its execution is described in 14.2.3, General rules of the procedure division; if intrinsic-function-name-1 is specified, its execution is described in 15, Intrinsic functions; if function-prototype-name-1 or function-pointer-name-1 is specified and the function to be activated is not a COBOL function, the execution is defined by the implementor.

e) After control is returned from the activated function, if an exception condition is propagated from the activated function, execution continues as specified in general rule 6f.

f) If an exception condition exists, any declarative that is associated with that exception condition is executed. Execution then proceeds as defined for the exception condition and execution of the declarative.

7) If the word OMITTED is specified or a trailing argument is omitted, the omitted-argument condition for that parameter evaluates to TRUE in the activated function. (See 8.8.4.1.7, Omitted-argument condition.)

8) If a parameter for which the omitted-argument condition is true is referenced in an activated function, except as an argument or in the omitted-argument condition, the EC-PROGRAM-ARG-OMITTED exception condition is set to exist and the results of the execution of the function are undefined.

8.4.2.3 Reference-modification

Reference modification defines a unique data item by specifying an identifier, a leftmost position, and a length.

8.4.2.3.1 General format

identifier-1(leftmost-position : [length])

8.4.2.3.2 Syntax rules

1) Identifier-1 shall reference a data item that is one of the following:

— a boolean data item,
— a national data item,

— an elementary data item of category alphanumeric or an alphanumeric group item,
— an alphabetic data item,
— a numeric-edited data item that is not subordinate to a strongly-typed group item,
— an alphanumeric-edited data item that is not subordinate to a strongly-typed group item,
— a national-edited data item that is not subordinate to a strongly-typed group item,
— a numeric data item of usage display or national that is not subordinate to a strongly-typed group item,
— a group item that is neither a strongly-typed group nor a variable-length group.

For purposes of reference modification, bit group items and national group items are treated as elementary data items.

2) If identifier-1 is a function-identifier, it shall reference an alphanumeric, boolean, or national function.

3) Identifier-1 shall not be a reference-modification format identifier.

4) Leftmost-position and length shall be arithmetic expressions.

5) Unless otherwise specified, reference modification is allowed anywhere an identifier referencing a data item of class alphanumeric, boolean, or national is permitted.

8.4.2.3.3 General rules

1) Leftmost-position represents a boolean position, alphanumeric position, or national position when identifier-1 references a boolean, alphanumeric, or national data item, respectively.

2) If the data item referenced by identifier-1 is explicitly or implicitly described as usage DISPLAY and its category is other than alphanumeric, identifier-1 is operated upon for purposes of reference modification as if it were redefined as a data item of class and category alphanumeric of the same size as the data item referenced by identifier-1.

3) If the data item referenced by identifier-1 is explicitly or implicitly described as usage NATIONAL and its category is other than national, it is operated upon for purposes of reference modification as if it were redefined as a data item of class and category national of the same size as the data item referenced by identifier-1.

4) Each position of the data item referenced by identifier-1 is assigned an ordinal number incrementing by one from the leftmost position to the rightmost position. The leftmost position is assigned the ordinal number one. If the data description entry for identifier-1 contains a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number within that data item.

5) Reference modification creates a unique data item that is a subset of the data item referenced by identifier-1. This unique data item is defined as follows:

   a) If the usage of identifier-1 is bit, positions used in evaluation are bit positions; otherwise, positions used in evaluation are character positions.

   b) The evaluation of leftmost-position specifies the ordinal position of the leftmost bit or character of the unique data item in relation to the leftmost bit or character of the data item referenced by identifier-1. Evaluation of leftmost-position shall result in a positive nonzero integer less than or equal to the number of positions in the data item referenced by identifier-1.

   c) The evaluation of length specifies the number of bit positions or character positions of the data item to be used in the operation. The evaluation of length shall result in a positive nonzero integer. The sum of leftmost-position and length minus the value one shall be less than or equal to the number of positions in the data item referenced by identifier-1. If length is not specified, the unique data item extends from and includes the position identified by leftmost-position up to and including the rightmost position of the data item referenced by identifier-1.

If the evaluation of leftmost-position or length results in a noninteger value or a value that references a position outside the area of identifier-1, the EC-BOUND-REF-MOD exception condition is set to exist.

NOTE When the runtime coded character set is the UTF-16 format of the UCS, the COBOL system does not detect reference modification that bisects the two halves of a surrogate pair.

6) The unique data item is considered to be an elementary data item without the JUSTIFIED clause. The unique data item has the same class, category, and usage as that defined for identifier-1, except that:

   a) the category alphanumeric-edited is considered class and category alphanumeric,

   b) the category national-edited is considered class and category national,

   c) the categories numeric and numeric-edited are considered class and category national if the usage is national; otherwise they are considered class and category alphanumeric, and

   d) an alphanumeric group item is considered to have usage display.

8.4.2.4 Inline method invocation

Inline method invocation references a temporary data item returned from invocation of a method.

8.4.2.4.1 General format

```
{ class-name-1 }
{ identifier-1 } :: literal-1 [ ( { arithmetic-expression-1 } ... ) ]
                              { boolean-expression-1      }
                              { identifier-2              }
                              { literal-2                 }
                              { OMITTED                   }
```

8.4.2.4.2 Syntax rules

1) Inline method invocation shall not be specified as a receiving operand.

2) Identifier-1 shall be of class object; neither the predefined object reference NULL nor a universal object reference shall be specified.

3) One of the INVOKE statements specified in general rule 1 shall be valid according to 14.9.22, INVOKE statement, syntax rules.

4) The data item referenced in the RETURNING phrase of the invoked method's procedure division header shall not be described with the ANY LENGTH clause or with the ACTIVE-CLASS phrase.

8.4.2.4.3 General rules

1) An inline method invocation references a temporary data item with the same class, category, and content as the temp-identifier that would be returned from the execution of the applicable form of INVOKE statement, as follows:

```
INVOKE identifier-1 literal-1 USING arguments RETURNING temp-identifier
INVOKE identifier-1 literal-1 RETURNING temp-identifier
INVOKE class-name-1 literal-1 USING arguments RETURNING temp-identifier
INVOKE class-name-1 literal-1 RETURNING temp-identifier
```

where:

a) arguments are the operands specified within parentheses in the inline method invocation, if any;

b) temp-identifier has the same description, class, and category as the RETURNING parameter in the specification of the method identified by literal-1 and either identifier-1 or class-name-1;

c) temp-identifier is a temporary item that exists for the purpose of effecting the inline invocation in this way and for no other purpose.

2) If an exception occurs during the execution of a statement containing this format, the resumption point is the next executable statement.

8.4.2.5 Object-view

An object-view causes an object reference to be treated as though it had the specified description. A runtime conformance check for this description will be done on the object.

8.4.2.5.1 General format

identifier-1 AS { [FACTORY OF] class-name-1 [ONLY] | interface-name-1 | UNIVERSAL }

8.4.2.5.2 Syntax rules

1) Identifier-1 shall be of class object; the predefined object references SUPER and NULL shall not be specified.

2) An object-view shall not be specified as a receiving operand.

8.4.2.5.3 General rules

1) This reference of identifier-1 is treated at compile-time as though it had the description specified by the AS phrase.

2) If class-name-1 is specified without either of the optional phrases, identifier-1 is treated as though it were described as USAGE IS OBJECT REFERENCE class-name-1. If the object referenced by identifier-1 is not an object of class-name-1 or an object of a subclass of class-name-1, the EC-OO-CONFORMANCE exception condition is set to exist.

3) If the FACTORY phrase is specified and the ONLY phrase is not specified, identifier-1 is treated as though it were described as USAGE OBJECT REFERENCE FACTORY OF class-name-1. If the object referenced by identifier-1 is not the factory object of class-name-1 or the factory object of a subclass of class-name-1, the EC-OO-CONFORMANCE exception condition is set to exist.

4) If the ONLY phrase is specified and the FACTORY phrase is not specified, identifier-1 is treated as though it were described as USAGE OBJECT REFERENCE class-name-1 ONLY. If the object referenced by identifier-1 is not an object of class-name-1, the EC-OO-CONFORMANCE exception condition is set to exist.

5) If both the FACTORY phrase and the ONLY phrase are specified, identifier-1 is treated as though it were described as USAGE OBJECT REFERENCE FACTORY OF class-name-1 ONLY. If the object referenced by identifier-1 is not the factory object of class-name-1, the EC-OO-CONFORMANCE exception condition is set to exist.

6) If interface-name-1 is specified, identifier-1 is treated as though it were described as USAGE OBJECT REFERENCE interface-name-1. If the object referenced by identifier-1 does not implement interface-name-1, the EC-OO-CONFORMANCE exception condition is set to exist.

7) If UNIVERSAL is specified, identifier-1 is treated as though it were described as USAGE OBJECT REFERENCE without any of the optional phrases to indicate the class or interface for objects referenced by identifier-1. The EC-OO-CONFORMANCE exception condition is not set to exist.

8.4.2.6 EXCEPTION-OBJECT

EXCEPTION-OBJECT is a predefined object reference that is used in a declarative procedure to reference the current exception object.

8.4.2.6.1 General format

EXCEPTION-OBJECT

8.4.2.6.2 Syntax rules

1) EXCEPTION-OBJECT shall not be specified as a receiving operand.

2) EXCEPTION-OBJECT is implicitly described as class object and category object reference, as an external data item, and as a universal object reference.

8.4.2.6.3 General rules

1) EXCEPTION-OBJECT references the current exception object. If an exception object is not associated with the current exception, EXCEPTION-OBJECT is set to null.

2) There is one instance of EXCEPTION-OBJECT in a run unit.

8.4.2.7 NULL object reference

NULL is a predefined object reference that contains the null object reference value.

8.4.2.7.1 General format

NULL

8.4.2.7.2 Syntax rules

1) NULL shall not be specified as a receiving operand.

2) NULL is implicitly described as class object and category object reference, and is not a universal object reference.

8.4.2.7.3 General rules

1) Predefined object reference NULL contains the null object reference value; this is a unique value defined by the implementor such that it is guaranteed to never reference an object.

8.4.2.8 SELF and SUPER

SELF and SUPER are predefined object references that reference the object on which the current method is executing.

8.4.2.8.1 General format

```
{ SELF                          }
{ [ class-name-1 OF ] SUPER     }
```

8.4.2.8.2 Syntax rules

1) This identifier format may be specified only in a method definition.

2) This identifier format shall not be specified as a receiving operand.

3) SUPER may be specified only as the object in an object-property identifier or as the object used to invoke a method with the INVOKE statement or an inline invocation of a method.

4) Class-name-1 shall be the name of a class specified in the INHERITS clause of the containing class definition.

5) If the INHERITS clause of the containing class definition specifies more than one class-name, class-name-1 shall be specified.

6) If the INHERITS clause of the containing class definition specifies only one class-name, class-name-1 may be specified.

7) SELF and SUPER are both implicitly described as class object and category object reference, and are not universal object references.

8.4.2.8.3 General rules

1) SELF and SUPER both reference the object that was used to invoke the method in which the reference to SELF or SUPER appears.

2) If SELF is specified for a method invocation, the method resolution is based upon the set of methods defined for the runtime class of the object referenced by SELF.

   NOTE The method resolution is not limited to the methods that are defined for the class that contains the method invocation. The object referenced by SELF at runtime may be an object of a subclass of the class that contains the invocation. Thus method invocation through the predefined object reference SELF uses the same method binding mechanism as is used for any other object identifier, based on the runtime class of the object.

3) If SUPER is specified for a method invocation, the method resolution ignores all the methods defined in the class containing the invocation and all the methods defined in any subclass of that class.

   NOTE The invoked method will be one that is defined in a superclass.

4) If class-name-1 is specified, the search for the method shall include only those methods defined for class-name-1.

8.4.2.9 Object property

Object properties provide a special syntax to get information out of and pass information back into an object. The mechanisms for accessing object properties are get property methods and set property methods. A get property method is a method explicitly defined with the GET PROPERTY phrase or a method implicitly generated for a data item described with the PROPERTY clause; a set property method is a method explicitly defined with the SET PROPERTY phrase or a method implicitly generated for a data item described with the PROPERTY clause.

8.4.2.9.1 General format

property-name-1 OF { class-name-1 / identifier-1 }

8.4.2.9.2 Syntax rules

1) Property-name-1 shall be an object property specified in the REPOSITORY paragraph.

2) Identifier-1 shall be an object reference; neither a universal object reference nor the predefined object reference NULL shall be specified.

3) If the object property is used as a sending item, a get property method shall exist for property-name-1 in the object referenced by identifier-1 or in the factory object of the class class-name-1.

4) If the object property is used as a receiving item, a set property method shall exist for property-name-1 in the object referenced by identifier-1 or in the factory object of the class class-name-1.

5) The description of an object property used as a sending item is the same as the description of the returning item of the get property method. This object property may be specified wherever a data item with that description would be valid as a sending item.

6) The description of an object property used as a receiving item is the same as the description of the using parameter of the set property method. This object property may be specified wherever a data item with that description would be valid as a receiving item.

7) The data description of the item specified in the RETURNING phrase of the get property method shall be the same as the data description of the item specified as the USING parameter of the set property method.

8.4.2.9.3 General rules

1) When an object property is used only as a sending item, a conceptual temporary data item, temp-1, is used in its place. The value of the property is determined as though the associated get property method were invoked, in accordance with the rules of the INVOKE statement, and the returned value placed in temp-1. The data description of temp-1 is the same as the data description of the item specified in the RETURNING phrase of the get property method.

2) When an object property is used only as a receiving item, a conceptual temporary data item, temp-2, is used in its place. The value of the property is assigned as though the associated set property method were invoked, in accordance with the rules of the INVOKE statement, passing the content of temp-2 as the parameter. The data description of temp-2 is the same as the data description of the item specified as the USING parameter of the set property method.

3) When an object property is used as both a sending item and a receiving item, conceptual temporary data items temp-1 and temp-2 are used in its place; temp-1 and temp-2 are the same temporary data item, where temp-2 redefines temp-1. For sending operations, the value of the property is determined in the same manner as for sending items in general rule 1; for receiving operations, the value of the property is assigned in the same manner as for receiving items in general rule 2. The data descriptions of temp-1 and temp-2 are the same as the data description of the item specified in the RETURNING phrase of the get property method.

8.4.2.10 NULL address pointer

NULL is a predefined address of class pointer.

8.4.2.10.1 General format

NULL

8.4.2.10.2 Syntax rules

1) This format may be used only as a sending operand in an INITIALIZE or a SET statement; as an argument in a program-prototype format CALL statement, a function-prototype format function activation, or a method invocation; or in a pointer-or-object-reference relation condition.

8.4.2.10.3 General rules

1) When associated with a data-pointer, the predefined address NULL references a data item of category data-pointer that contains the null address. The null data address is an implementor-defined value that is guaranteed not to represent the address of any data item.

2) When associated with a function-pointer, the predefined address NULL references a data item of category function-pointer that contains the NULL function address. The null function address is an implementor-defined value that is guaranteed not to represent the address of any function.

3) When associated with a program-pointer, the predefined address NULL references a data item of category program-pointer that contains the NULL program address. The null program address is an implementor-defined value that is guaranteed not to represent the address of any program.

8.4.2.11 Data-address-identifier

A data-address-identifier references the unique data item that contains the address of a data item.

8.4.2.11.1 General Format

ADDRESS OF identifier-1

8.4.2.11.2 Syntax rules

1) Identifier-1 shall reference a data item defined in the file section, working-storage section, local-storage section, or linkage section. Identifier-1 shall not be defined in the working-storage or file section of an object or a factory object.

2) Identifier-1 shall not reference an object reference or an elementary item subordinate to a strongly-typed group item.

3) Identifier-1 shall not reference a data item that is described with the CONSTANT RECORD clause, or any data item subordinate to such a data item.

4) If identifier-1 is a bit data item, identifier-1 shall be described such that:

   a) subscripting and reference modification in identifier-1 consist of only fixed-point numeric literals or arithmetic expressions in which all operands are fixed-point numeric literals and the exponentiation operator is not specified; and

   b) it is aligned on a byte boundary.

5) If identifier-1 references a strongly-typed group item, identifier-1 shall be defined with a TYPE clause referencing a type declaration described with the STRONG phrase.

   NOTE The effect of this rule, along with general rule 2, is to produce a restricted data-pointer.

6) This identifier format shall not be specified as a receiving operand.

7) Identifier-1 shall not reference an any-length elementary item, an element of a dynamic-capacity table, an item subordinate to a dynamic-capacity table, or an item subordinate to a group that contains an any-length elementary item.

8.4.2.11.3 General rules

1) Data-address-identifier creates a unique data item of class pointer and category data-pointer that contains the address of identifier-1.

2) If identifier-1 is a strongly-typed group item or a restricted data-pointer, the data-address-identifier is a restricted data pointer that is restricted to the type of identifier-1.

8.4.2.12 Function-address-identifier

A function-address-identifier identifies the unique data item that contains the address of a function.

8.4.2.12.1 General Format

ADDRESS OF FUNCTION { function-prototype-name-1 | identifier-1 }

8.4.2.12.2 Syntax rules

1) Identifier-1 shall be of category alphanumeric or national.

2) Function-prototype-name-1 shall be a function prototype specified in the REPOSITORY paragraph.

3) This identifier format shall not be specified as a receiving operand.

8.4.2.12.3 General rules

1) Function-address-identifier creates a unique data item of class pointer and category function-pointer that contains the address of a function identified by one of the following:

   a) the content of the data item referenced by identifier-1

   b) function-prototype-name-1.

   If identifier-1 is specified, paragraph 8.3.1.1.1, User-defined words, describes how this value is used to identify the referenced function. Identifier-1 shall reference a function prototype specified in the REPOSITORY paragraph.

2) The function may be written in COBOL or in another language for which the implementor has declared support. For a COBOL function, the address is that of the function identified by the externalized function-name in its FUNCTION-ID paragraph. For a non-COBOL function, the relation between the address and the associated function is defined by the implementor.

3) When function-prototype-name-1 is specified, the function-address-identifier has the characteristics of a function-pointer restricted to function-prototype-name-1.

4) If the runtime system cannot locate the function, the EC-FUNCTION-NOT-FOUND exception condition is set to exist and the value of the address-identifier is the predefined address NULL.

8.4.2.13 Program-address-identifier

A program-address-identifier references the unique data item that contains the address of a program.

8.4.2.13.1 General Format

ADDRESS OF PROGRAM { identifier-1 | literal-1 | program-prototype-name-1 }

8.4.2.13.2 Syntax rules

1) Identifier-1 shall be of category alphanumeric or national.

2) Literal-1 shall be an alphanumeric or national literal whose length is not zero.

3) Program-prototype-name-1 shall be a program prototype specified in the REPOSITORY paragraph.

4) This identifier format shall not be specified as a receiving operand.

8.4.2.13.3 General rules

1) Program-address-identifier creates a unique data item of class pointer and category program-pointer that contains the address of a program identified by one of the following:

   a) the content of the data item referenced by identifier-1

   b) the value of literal-1

   c) program-prototype-name-1.

   If identifier-1 or literal-1 is specified, paragraph 8.3.1.1.1, User-defined words, describes how this value is used to identify the referenced program.

2) The program may be written in COBOL or in another language for which the implementor has declared support. For a COBOL program, the address is that of the outermost program identified by the externalized program-name in its PROGRAM-ID paragraph. For a non-COBOL program, the relation between the address and the associated program is defined by the implementor.

3) When program-prototype-name-1 is specified, the program-address-identifier has the characteristics of a program-pointer restricted to program-prototype-name-1.

4) If the runtime system cannot locate the program, the EC-PROGRAM-NOT-FOUND exception condition is set to exist and the value of the address-identifier is the predefined address NULL.

8.4.2.14 LINAGE-COUNTER

The LINAGE-COUNTER identifier is generated by the presence of a LINAGE clause in a file description entry.

8.4.2.14.1 General format

LINAGE-COUNTER [{ IN | OF } file-name-1]

8.4.2.14.2 Syntax rules

1) LINAGE-COUNTER may be referenced only in procedure division statements.

2) The LINAGE-COUNTER identifier shall not be referenced as a receiving operand.

3) Qualification requirements for LINAGE-COUNTER are defined by 8.4.1.1, Qualification.

8.4.2.14.3 General rules

1) LINAGE-COUNTER references a temporary unsigned integer data item of class and category numeric whose size is equal to the page size specified in the LINAGE clause.

2) The semantics of the LINAGE-COUNTER identifier is described in 13.18.33, LINAGE clause, general rule 7.

8.4.2.15 Report counters

The PAGE-COUNTER and LINE-COUNTER identifiers are generated automatically and exist independently for each report.

8.4.2.15.1 General format

```
{ PAGE-COUNTER }  [ { IN }           ]
{ LINE-COUNTER }  [ { OF } report-name-1 ]
```

8.4.2.15.2 Syntax rules

1) In the report section, PAGE-COUNTER and LINE-COUNTER may be referenced only in a SOURCE clause. In the procedure division, PAGE-COUNTER and LINE-COUNTER may be referenced in any context where an integer data item may appear.

2) Qualification requirements for PAGE-COUNTER and LINE-COUNTER are defined by 8.4.1.1, Qualification.

> NOTE Because each report maintains an independent PAGE-COUNTER and LINE-COUNTER, it is the programmer's responsibility to assign the correct values to any page numbers and to ensure that report groups are printed correctly within the limits of the page.

3) LINE-COUNTER shall not be referenced as a receiving operand.

8.4.2.15.3 General rules

1) PAGE-COUNTER and LINE-COUNTER reference temporary unsigned integer data items of class and category numeric, which are maintained for each report.

2) The initial value of PAGE-COUNTER is set to 1 by the execution of an INITIATE statement for the corresponding report and its value is updated by 1 during each page advance. It is reset to 1 when a report group that contains a NEXT GROUP clause with a RESET phrase is printed.

3) The initial value of LINE-COUNTER is set to zero by execution of an INITIATE statement for the corresponding report. It is reset to zero whenever a page advance takes place.

4) At the time each report line is printed, the value of LINE-COUNTER specifies the line number of the page on which the line is printed. The value of LINE-COUNTER after the printing of a report group is the same as the line number of the last line printed, unless a NEXT GROUP clause is defined for the report group, in which case the final value of LINE-COUNTER is defined by the general rules for the NEXT GROUP clause.

5) The values of PAGE-COUNTER and LINE-COUNTER are not affected by the processing of a dummy report group, nor by the processing of a report group whose printing is suppressed by means of the SUPPRESS statement.

8.4.3 Condition-name

There are two kinds of condition-names. One is used to identify a subset of the values that an associated data item may assume. The other is associated with the on status or off status of an implementor-defined switch.

The level-number 88 identifies a condition-name and a specific value, set of values, or range of values. This condition-name is associated with the data item to which it is subordinate, called a conditional variable. Referencing this condition-name in a condition, as described in 8.8.4.1.4, Condition-name condition (conditional variable), is an abbreviation for the conditional expression that posits that the value of the associated conditional variable is equal to one of the set of values identified with condition-name. This kind of condition-name is defined in 13.16, Data description entry. This kind of condition-name can be used in a SET statement to move a value to the associated conditional variable to make the condition-name either 'true' or 'false'.

Within the SPECIAL-NAMES paragraph, a condition-name identifies the on status or off status of an implementor-defined switch. Referencing this condition-name, as described in 8.8.4.1.5, Switch-status condition, in a condition posits that the associated switch has the 'on' or 'off' status that is associated with the condition-name. This condition-name can also be used in a SET statement to set the associated switch to the 'on' or 'off' status.

8.4.3.1 General format

Format 1 (switch-status-condition-name):

condition-name-1

Format 2 (qualified-condition-name-with-subscripts):

qualified-condition-name-with-subscripts-1

8.4.3.2 Syntax rules

FORMAT 1

1) Condition-name-1 shall be associated with a switch-name in the SPECIAL-NAMES paragraph.

FORMAT 2

2) Qualified-condition-name-with-subscripts-1 is defined by 8.4.1.1, Qualification.

8.4.4 Explicit and implicit references

A source element may reference data items either explicitly or implicitly in procedure division statements. An explicit reference occurs when the name of the referenced item is written in a procedure division statement. An implicit reference occurs when the item is referenced by a procedure division statement without the name of the referenced item being written in the source statement. An implicit reference also occurs, during the execution of a PERFORM statement, when the index or data item referenced by the index-name or identifier specified in the VARYING, AFTER, or UNTIL phrase is initialized, modified, or evaluated by the control mechanism associated with that PERFORM statement. Such an implicit reference occurs if and only if the data item contributes to the execution of the statement.

8.4.5 Scope of names

When source elements are directly or indirectly contained within other source elements, each source element may use identical user-defined words to name items independent of the use of these user-defined words by other

source elements. (See 8.3.1.1.1, User-defined words.) When identically named items exist, a source element's reference to such a name, even when it is a different type of user-defined word, is to the item which that source element describes rather than to the item, possessing the same name, described in another source element.

Except for program-names and method-names, a source element shall not reference any name declared in any source element it contains.

The following types of user-defined words may be referenced throughout a compilation group:

- library-name
- text-name

The following types of user-defined words may be referenced only by statements in the source element in which the user-defined word is declared:

- paragraph-name
- section-name

The following types of user-defined words, when they are declared within a configuration section, may be referenced only by statements and entries either in that source element that contains a configuration section or in any source element contained within that source element:

- alphabet-name
- class-name (for truth value proposition)
- condition-name (declared in a configuration section)
- locale-name
- mnemonic-name
- ordering-name
- symbolic-character

Specific conventions for declarations and references to the following types of user-defined words are specified in 8.4.5.1, Local and global names, through 8.4.5.10, Scope of property-names:

- class-name (for object orientation)
- compilation-variable-name
- condition-name (not declared in a configuration section)
- constant-name
- data-name
- file-name
- function-prototype-name
- index-name
- interface-name
- method-name
- parameter-name
- program-name
- program-prototype-name
- property-name
- record-key-name
- record-name
- report-name
- screen-name
- type-name
- user-function-name

8.4.5.1 Local and global names

A local name may be referenced only in the source element in which it is declared.

A global name may be referenced in the source element in which it is declared or in any source elements that are directly or indirectly contained within that source element.

When a source element, source element B, is directly contained within another source element, source element A, both source elements may define a name using the same user-defined word. In addition, source element A may be contained in another source element and that other source element may also define a name using the same user-defined word. When such a duplicated name is referenced in source element B, the following rules are used to determine the referenced item:

1) The set of names to be used for determination of a referenced item consists of all names that are defined in source element B and all global names that are defined in source element A and in any source elements that directly or indirectly contain source element A. Using this set of names, the normal rules for qualification and any other rules for uniqueness of reference are applied until one or more items is identified.

2) If only one item is identified, it is the referenced item.

3) If more than one item is identified, no more than one of them may have a name local to source element B. If zero or one of the items has a name local to source element B, the following rules apply:

    a) If the name is declared in source element B, the item in source element B is the referenced item.

    b) Otherwise, if source element A is contained within another source element, the referenced item is:

        1. The item in source element A if the name is declared in source element A.

        2. The item in the containing source element if the name is not declared in source element A and is declared in the source element containing source element A. This rule is applied to further containing source elements until a single valid name has been found.

8.4.5.1.1 Scope of condition-names, constant-names, data-names, file-names, record-names, report-names, screen-names, and type-names

A constant-name, file-name, record-name, report-name, screen-name, or type-name described with a GLOBAL clause is a global name. A constant-name, data-name, file-name, or type-name declared in a source element for an object definition, whether factory or instance, is a global name. All data-names and screen-names subordinate to a global name are global names. All condition-names associated with a global name are global names.

When a condition-name, constant-name, data-name, file-name, record-name, report-name, screen-name, or type-name is not a global name, it is a local name.

The requirements governing the uniqueness of the names allocated by a single source element to be condition-names, constant-names, data-names, file-names, record-names, report-names, screen-names, and type-names are explained elsewhere in these specifications. (See 8.3.1.1.1, User-defined words.)

8.4.5.1.2 Scope of index-names

If a data item possessing the global attribute includes a table described with an index-name, that index-name also possesses the global attribute. Therefore, the scope of an index-name is identical to that of the data-name that names the table whose index is named by that index-name.

8.4.5.1.3 Scope of record-key-names

The record-key-name defined by a SOURCE phrase in the ALTERNATE RECORD KEY clause or RECORD KEY clause of the file control entry for an indexed file is global if the GLOBAL clause is specified in the file description entry for that file; otherwise, the record-key-name is local.

8.4.5.1.4 Scope of PAGE-COUNTER AND LINE-COUNTER

PAGE-COUNTER and LINE-COUNTER are global if the GLOBAL clause is specified in report description entry of the associated report; otherwise, they are local.

8.4.5.1.5 Scope of LINAGE-COUNTER

LINAGE-COUNTER is global if the GLOBAL clause is specified in the file description entry for the associated file; otherwise, it is local.

8.4.5.2 Scope of program-names

The program-name of a program is declared in the PROGRAM-ID paragraph of the program's identification division. A program-name may be referenced only by the CALL statement, the CANCEL statement, the program-address-identifier, and the end program marker. The names assigned to programs that are contained directly or indirectly within the same outermost program shall be unique within that outermost program.

The following rules regulate the scope of a program-name for the CALL and CANCEL statements, and the program-address-identifier:

1) If the program-name is that of a program that does not possess the common attribute and that is directly contained within another program, that program-name may be referenced only by statements included in that containing program or, if the program possesses the recursive attribute, in the program itself.

2) If the program-name is that of a program that does possess the common attribute and that is directly contained within another program, that program-name may be referenced only by statements included in that containing program and any programs directly or indirectly contained within that containing program, except that the program possessing the common attribute and any programs contained within it may reference the program-name only if the program possesses the recursive attribute.

3) If the program-name is that of an outermost program, that program-name may be referenced by statements included in any source element in the run unit.

8.4.5.3 Scope of class-names and interface-names

The class-name of a class referenced within a source element shall be either the name of the containing class definition or declared in the REPOSITORY paragraph of that or a containing source element.

Class definitions within a compilation group shall have unique class-names.

The interface-name of an interface referenced within a source element shall be either the name of the containing interface definition or declared in the REPOSITORY paragraph of that or a containing source element.

Interface definitions within a compilation group shall have unique interface-names.

A class-name or interface-name declared in the REPOSITORY paragraph of a source element may be used in that source element and any nested source element.

8.4.5.4 Scope of method-names

The method-name of a method is declared in the METHOD-ID paragraph. A method-name may be referenced only by the INVOKE statement, an inline invocation, and the end method marker.

The methods declared in a class definition shall have unique method resolution signatures within that class definition. The methods declared in a subclass may have the same method resolution signature as a method in the superclass, subject to the conditions in 11.7, METHOD-ID paragraph.

The methods declared in an interface definition shall have unique method resolution signatures within that interface definition. The methods declared in an inheriting interface may have the same method resolution signature as a method in the inherited interface, subject to the conditions in 11.7, METHOD-ID paragraph.

8.4.5.5 Scope of function-prototype-names

Function-prototype-names referenced within a source element shall be either the user-function-name of the containing function definition or a function-prototype-name declared in the REPOSITORY paragraph.

8.4.5.6 Scope of user-function-names

A user-function-name may be referenced in the REPOSITORY paragraph of any source element that follows that function definition within the compilation group and, if the external repository is updated, in any subsequently-compiled source unit that specifies that user-function-name as a function-prototype-name in its REPOSITORY paragraph.

8.4.5.7 Scope of program-prototype-names

Program-prototype-names referenced within a source element shall be either the program-name of a containing program definition or a program-prototype-name declared in the REPOSITORY paragraph

8.4.5.8 Scope of compilation-variable-names

The scope of a compilation-variable-name is from the point of definition to the end of the compilation group. Compilation-variable-names may be referenced in compiler directives and in a constant entry.

8.4.5.9 Scope of parameter-names

Parameter-names may be referenced only within the class definition or interface definition in which they are specified in the USING phrase, subject to the rules in 11.3, CLASS-ID paragraph, or 11.6, INTERFACE-ID paragraph.

8.4.5.10 Scope of property-names

The property-name of a property referenced within a source element shall be declared in the REPOSITORY paragraph of that or a containing source element.

8.5 Data description and representation

8.5.1 Computer independent data description

To make data as computer-independent as possible, the characteristics or properties of data are described in the data division in a format that is largely independent from the manner in which data are stored internally in the computer or on a particular external medium. When the implementation provides multiple ways of storing data, the clauses of the data description entries determine the specific representation of the data in storage. Each implementor shall provide a complete specification of the possible representations on the computer for which COBOL is implemented, except when specific reference is made to other standards defining such representations.

Except for the hexadecimal formats, the contents of literals are described in the computer's coded character set known at compile time. When a different coded character set is in effect on the computer at runtime, the content of the literal is converted to the computer's runtime coded character set as described in 8.1.1, Computer's coded character set. The hexadecimal literal formats specify the bit patterns to be used at runtime.

8.5.1.1 Files and records

COBOL has some language elements for describing physical aspects of a file, like the external name used to associate the logical file with a physical file, and the grouping of logical records within the physical limitations of the file medium.

For the most part, COBOL deals with logical files. COBOL input and output statements refer to logical records. Each logical record consists of a set of data description entries that describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name, if required, followed by a series of independent clauses, as required.

Data items described in the working-storage section, the local storage section, or the linkage section can also be grouped into logical records using record description entries.

Files, including their physical aspects, the relationship between physical and logical files, and the characteristics of logical files, are described in 9.1, Files.

8.5.1.2 Levels

A level concept is inherent in the structure of a record. This concept arises from the need to specify subdivision of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referral.

The most basic subdivisions of a record, that is, those not further subdivided, are called elementary items; consequently, a record is said to consist of a sequence of elementary items, or the record itself may be an elementary item.

In order to refer to a set of elementary items, the elementary items are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into groups of one or more groups. An elementary item may belong to more than one group in a hierarchy of groups.

8.5.1.2.1 Level-numbers

A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 1. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 49. There are special level-numbers, 66, 77, and 88, that are exceptions to this rule.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. All items that are immediately subordinate to a given group item shall be described using numerically equal level-numbers greater than the level-number used to describe that group item.

Three types of entries exist for which there is no true concept of level. These are:

1) Entries that specify elementary items or groups introduced by a RENAMES clause.

2) Entries that specify noncontiguous working-storage, local storage, and linkage data items.

3) Entries that specify condition-names.

Entries describing items by means of RENAMES clauses for the purpose of re-grouping data items have been assigned the special level-number 66.

Entries that specify noncontiguous data items that are not subdivisions of other items, and are not themselves subdivided, have been assigned the special level-number 77.

Entries that specify condition-names to be associated with particular values of a conditional variable have been assigned the special level-number 88.

8.5.1.3 Limitations of character handling

Each coded character of the character sets supported by an implementation is processed at runtime as a single character position. The following processing limitations apply when ISO/IEC 10646 is chosen as a computer's coded character set:

1) the noncombining character and the following combining characters of a composite sequence defined in ISO/IEC 10646 are each treated as a single character position.

2) the R-octet (high-half 2 octets) and the C-octet (low-half 2 octets) of a four-octet sequence defined in the UTF-16 format of ISO/IEC 10646 are each treated as a single character position.

NOTE The UTF-16 format of ISO/IEC 10646 is a coded character set where each code element (or 'code value') consists of two octets. Many, but not all, of the letters and symbols are represented in one code element. In order to accommodate more entities than could otherwise be defined in a two-octet coded character set, the following techniques are used in ISO/IEC 10646:

— some abstract characters or text entities are defined as 'combining sequences' of multiple code elements where a two-octet base character and one or more two-octet combining characters form a complete entity, together called a composite sequence.

— some abstract characters or text entities are defined as 'surrogate pairs' consisting of two code elements that form a high-half and a low-half of the abstract character or text entity.

These techniques facilitate efficient processing of data as fixed-size two-octet code elements, but present opportunities for corruption of data if not handled correctly. To achieve a better understanding, see ISO/IEC 10646

COBOL supports UTF-16 as a Level U implementation as defined in ISO/IEC 10646, which means that COBOL does not provide any special handling or recognition of surrogate pairs; nor does COBOL provide recognition of composite sequences. Each two-octet code element of UTF-16 is treated in COBOL as though it were itself a character. Users are responsible for ensuring that any truncation or replacement that occurs is consistent with the needs of their application.

8.5.1.4 Algebraic signs

Algebraic signs fall into two categories: operational signs, which are associated with signed numeric data items and signed numeric literals to indicate their algebraic properties; and editing signs, which appear in edited data items to identify the sign of the item.

The SIGN clause permits the programmer to state explicitly the location of the operational sign. This clause is optional; if it is not used, operational signs will be represented as defined by the implementor.

Editing signs are inserted into a data item through the use of the sign control symbols of the PICTURE clause.

8.5.1.5 Alignment of data items in storage

8.5.1.5.1 Alignment of alphanumeric groups and of data items of usage display

Alignment of alphanumeric groups and of data items of usage display is at a natural alphanumeric character boundary and is coincident with a byte boundary in the architecture of the processor.

The alignment of the start of an alphanumeric group item relative to the first item within that group is defined by the implementor, with the exception that alignment of the group is coincident with the first item when that item is a national data item or a data item of class boolean, object, or pointer.

8.5.1.5.2 Alignment of data items of usage national

Alignment of data items of usage national is at a natural national character boundary.

The alignment of the start of a group item and the alignment of the start of the first item within that group, when the first item is of usage national, are at the same position in storage.

8.5.1.5.3 Alignment of data items of usage bit

Alignment of elementary bit data items and bit group items within a record, when neither a SYNCHRONIZED clause nor an ALIGNED clause is specified, is at the next bit position in storage when that item is:

— an elementary bit data item immediately following an elementary bit data item or bit group item of the same level;

— a bit group item immediately following a bit group item or elementary bit data item of the same level.

Alignment of all other bit data items within a record, when a SYNCHRONIZED clause is not specified, is at the first bit position of the first available byte.

Alignment of elementary bit data items of level 1 or level 77 and a level 1 bit group are at the first bit of a byte.

Implicit filler bit positions are generated:

— As defined by the implementor for a bit data item described with the SYNCHRONIZED clause. The implementor defines the positioning rules associated with any filler bit positions.

— Following a bit data item within an alphanumeric group item, within a strongly-typed group item, or within a bit group item, as needed to advance alignment to a required natural boundary for the next item within that group. The filler bit positions are implicitly described as a filler elementary bit data item of the necessary number of bits and of the same level number as the next item within that group.

— Following a bit data item that is the last data item in a record that is an alphanumeric group or strongly-typed group item, as needed to increase the number of bits to fill an integral number of characters. The filler bit positions are implicitly described as a filler elementary bit data item of the necessary number of bits with a level number the same as the highest hierarchical level of any bit data item superordinate to the last item, or, if there is no such superordinate item, the same as the level number of the last data item in the record.

NOTE No filler is generated at the end of a record that is entirely a bit group, at the end of a level 77 item, or at the end of a level 1 elementary item.

The alignment of the start of a group item and the alignment of the first item within that group, when the first item is a bit data item, are at the same bit position in storage.

8.5.1.5.4 Item alignment for increased object-code efficiency

Some computer memories are organized in such a way that there are natural addressing boundaries in the computer memory, such as word boundaries, half-word boundaries, and byte boundaries. The way in which data is stored is determined by the runtime module, and need not respect these natural boundaries.

However, certain uses of data in such constructs as arithmetic operations or subscripting may be facilitated if the data is stored so as to be aligned on these natural boundaries. Specifically, additional operations might be required at runtime for the accessing and storage of data if portions of two or more data items appear between adjacent natural boundaries, or if certain natural boundaries bifurcate a single data item.

Data items that are aligned on these natural boundaries in such a way as to avoid such additional machine operations are defined to be synchronized.

Synchronization is accomplished in two ways:

1) By use of the SYNCHRONIZED clause.

2) By recognizing the appropriate natural boundaries and organizing the data suitably without the use of the SYNCHRONIZED clause.

Each implementor who provides for special types of alignment shall specify the precise interpretations that are to be made. The use of such items within a group may affect the results of statements in which the group is used as an operand. Each implementor who provides for these special types of alignment shall describe the effect of the implicit FILLER and the semantics of any statement referencing these groups.

8.5.1.5.5 Alignment of strongly-typed group items

Alignment of strongly-typed group items is at a natural alphanumeric character boundary and is coincident with a byte boundary in the architecture of the processor.

The alignment of the start of a strongly-typed group item relative to the first item within that group is defined by the implementor, with the following exceptions:

— The alignment of the group shall be coincident with the first item when that item is a national group item or a data item of class boolean, object, or pointer.

— The alignment of the elementary items contained in the group shall be such that the essential characteristics of the type, as defined in 8.5.3, Types, is preserved.

8.5.1.6 Tables

Repetitive data may be described in COBOL as a table. Tables may have multiple dimensions. Elements of tables may be elementary items or groups. Elements of a table are referenced using subscripts, as described in 8.4.1.2, Subscripts. COBOL provides for three types of tables: fixed-capacity tables, occurs-depending tables, and dynamic-capacity tables. Additional rules for defining and using tables may be found in 13.18.37, OCCURS clause.

The capacity of a table is the number of occurrences of the elements of that table. The logical capacity of a table is the number of occurrences whose content is defined at any given time in accordance with general rule 7 of 13.18.37, OCCURS clause. The physical capacity of that table is the number of occurrences for which physical resources have been allocated.

8.5.1.6.1 Fixed-capacity tables

Fixed-capacity tables are tables whose physical and logical capacities are the same and are fixed at compile time. Fixed-capacity tables are defined by an OCCURS clause without either the DEPENDING phrase or the DYNAMIC phrase.

8.5.1.6.2 Occurs-depending tables

Occurs-depending tables are tables whose physical capacity is fixed at compile time; however, the logical capacity may vary during execution. The range of the logical capacity is specified in the DEPENDING phrase of the OCCURS clause.

8.5.1.6.3 Dynamic-capacity tables

Dynamic-capacity tables are tables whose physical and logical capacities may vary during execution. The logical capacity of a dynamic-capacity table is equal to its physical capacity. The number of occurrences allocated at a particular time is referred to as the current capacity of the table. A dynamic-capacity table differs from an occurs-depending table in that:

1) its occurrences are created dynamically,

2) it may have an unlimited capacity, and

3) it may be defined in any place, other than the file section, in which a fixed-capacity table may be defined and may be nested in any combination to the same number of levels as a fixed-capacity table.

A dynamic-capacity table also differs from fixed-capacity and occurs-depending tables in that table data items are not required to be contiguous in physical memory. The size, location, and time of physical allocation of a dynamic-capacity table are implementor-defined.

The current capacity of a dynamic-capacity table may be initialized explicitly in the FROM phrase of the OCCURS clause or implicitly in the VALUE clause. If neither is specified, the current capacity is initialized to zero.

The TO phrase of the OCCURS clause may be used to specify an upper value for the current capacity of a dynamic table, which may be exceeded with a nonfatal exception. This value is referred to as the expected capacity. The actual limit for the current capacity imposed by the implementor and by current resource availability is referred to as the maximum capacity.

8.5.1.6.3.1 Operations on a single element

A data item in a dynamic-capacity table is referenced using a subscript, in the same way as specified for a fixed-capacity table in 8.4.1.2, Subscripts.

When a data item in a dynamic-capacity table is referenced as a sending operand, the result of the operation is the same as for a fixed-capacity table whose number of occurrences is the current capacity of the table.

When a data item in a dynamic-capacity table is referenced as a receiving item and the value of the subscript does not exceed the current capacity of the table, the result of the operation is the same as for a fixed-capacity table whose number of occurrences is the current capacity of the table.

8.5.1.6.3.2 Implicit changes in capacity

When a data item in a dynamic-capacity table is referenced as a receiving item and the value of the subscript exceeds the current capacity of the table, a new element is automatically created and the capacity of the table is increased to the value given by the subscript. If this new capacity is more than one greater than the previous capacity, new intermediate occurrences are implicitly created.

NOTE There is no requirement for neighboring occurrences in a dynamic-capacity table to be physically contiguous with each other.

8.5.1.6.3.3 Explicit changes in capacity

If the OCCURS clause specifies a CAPACITY phrase, the capacity of the dynamic-capacity table may be increased or decreased explicitly by means of the dynamic-capacity-table format SET statement. If the capacity of the table

is thereby reduced, the appropriate number of higher occurrences is deleted and any resources they were using are freed.

NOTE The implementor may keep the resources assigned to a table for efficiency's sake and release them only when they are required, provided that the functionality of the program is not thereby affected in any way.

8.5.1.6.3.4 Implicit initialization

If the INITIALIZED phrase is specified in the OCCURS clause of a dynamic-capacity table, any elementary items not referenced as receiving operands in a statement that creates new entries in that table are first initialized as though they had been the subject of a statement of the form INITIALIZE … WITH FILLER ALL TO VALUE THEN TO DEFAULT. Otherwise, the content of any unreferenced locations in the new entry and the content of the other new occurrences are undefined.

NOTE The implementor need not physically allocate and, if specified, initialize such intermediate occurrences, provided that at runtime the implementation of the program is functionally identical to one where the intermediate occurrences are physically allocated and, if specified, initialized.

8.5.1.6.3.5 Operations on entire tables

Operations may be performed on an entire dynamic table as a result of an operation on a group that contains it. If another operand of the operation is a fixed-capacity or occurs-depending table, that table shall be treated as a special case of a dynamic-capacity table having a current capacity equal to the fixed number of entries or the value of the corresponding DEPENDING operand, as respectively.

8.5.1.6.3.5.1 Moving a table

A dynamic-capacity table may be moved as part of a variable-length group to or from another table, whether dynamic-capacity, occurs-depending, or fixed-capacity, defined in a compatible group as specified in 8.5.1.9, Variable-length groups. The operation recreates or overwrites the receiving table with a copy of the sending table, after freeing, if applicable, all the resources previously occupied by the receiving table.

If the receiving table is not a dynamic-capacity table:

1) If the sending table has a higher current capacity than the receiving table, superfluous entries are not moved and no exception exists,

2) If the sending table has a lower current capacity than the receiving table, all the remaining entries of the receiving table are space filled.

If the receiving table is a dynamic-capacity table specifying a minimum capacity that is higher than its current capacity, further entries are created and filled with spaces until the current capacity of the table is equal to its minimum capacity as specified in 8.5.1.6.3.5.3, Space filling a dynamic table.

Correspondingly numbered entries are moved according to the rules of the MOVE statement specified in 14.9.24, MOVE statement.

8.5.1.6.3.5.2 Comparing two tables

During the evaluation of a relation condition, a dynamic-capacity table may be compared with a another dynamic-capacity table, an occurs-depending table, or a fixed-capacity table, either directly or subordinate to the comparison of two compatible groups as specified in 8.5.1.9, Variable-length groups.

When corresponding tables are to be compared, the first element of the first table, if any, is compared with the first element of the other table. If they are unequal, the comparison terminates. Otherwise, the next elements are compared, continuing until either inequality is detected or the last element of the table with the smallest current capacity has been compared. If the two tables do not have the same current capacity, comparison continues by comparing each successive remaining entry of the larger table with spaces.

8.5.1.6.3.5.3 Space filling a dynamic table

If a dynamic table is subordinate to a variable-length group that is to be space filled as part of the execution of a MOVE statement, each entry of the dynamic table is space filled and the current capacity of the dynamic table is unaffected.

8.5.1.6.3.6 Exceeding capacity

The creation of a new entry in a dynamic-capacity table can result in one of the following exceptions:

1) EC-BOUND-OVERFLOW. The nonfatal EC-BOUND-OVERFLOW exception condition shall exist when a dynamic-capacity table has an expected capacity and an operation causes this expected capacity to be exceeded. If the change in capacity was implicit and the expected capacity had already been exceeded before the operation, no exception shall exist.

   If checking for the EC-BOUND-OVERFLOW exception condition is not turned on, or results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, the operation shall be allowed to continue, thus exceeding the receiving table's specified expected capacity.

2) EC-BOUND-TABLE-LIMIT. The fatal EC-BOUND-TABLE-LIMIT exception condition shall exist when an operation attempts to increase the capacity of a dynamic-capacity table to a value higher than the maximum capacity of the table based on the resources available at runtime.

8.5.1.7 Any-length elementary items

An any-length elementary item is defined by the ANY LENGTH clause. An any-length elementary item is one for which the number of character positions allocated to the data item may vary during program execution. The current length of the data item is the number of character positions in the content of the data item at any given time. An any-length elementary item may be category alphanumeric, national or boolean.

The maximum size of an any-length elementary item is smallest of:

- the value declared in the LIMIT phrase
- the largest integer that can be stored in an item of the usage specified in the PREFIXED phrase
- the maximum permitted by the implementor

An elementary data item that is not variable-length is referred to as a fixed-length data item.

8.5.1.7.1 Structure of an any-length elementary item

The internal structure of an any-length elementary item is defined by reference to an any-length-structure-name. The any-length-structure-name, defined in the ANY LENGTH STRUCTURE clause in the SPECIAL-NAMES paragraph, may refer either to an explicit definition, using the PREFIXED and DELIMITED clauses, or to an implementor-defined definition. In the absence of an any-length-structure-name the internal structure is defined by the implementor.

8.5.1.7.2 Location of any-length elementary items

Any-length elementary items may be physically located in memory within the record they are subordinate to, or they may be located elsewhere in the computer's memory. The actual physical location is defined by the implementor, and may change at any time during execution.

8.5.1.7.3 Operations on any-length elementary items

An any-length elementary item that is used as a sending operand or is reference-modified is treated as a fixed-length data item whose length is the any-length elementary item's current length.

If an any-length elementary item is a receiving operand and is not reference-modified, the new value becomes the content of the item. The new length of the any-length elementary item is determined by the length of new content. If the length of the sending operand is zero, no data is moved and the new length of the any-length elementary item is set to zero.

If the maximum length is reached, the value is truncated on the right as necessary.

8.5.1.8 Variable-length data items

The term variable-length data item refers to either a dynamic-capacity table or an any-length elementary item.

8.5.1.8.1 Contiguity of data items

A variable-length data item may be part of any group structure, and its neighbors may be non-variable-length data items or variable-length data items. A variable-length data item is logically but not necessarily physically contiguous with its neighbors. However a variable-length data item behaves in all respects as though it were in fact contiguous with its neighbors whenever a procedural operation is applied to a group containing it.

The physical address of a variable-length data item may change during execution of the program. Dynamic-capacity tables and indirect any-length elementary items, however they may change during execution, do not in any way affect the addresses of their neighbors.

8.5.1.8.2 Availability and persistence of data items

The availability of a variable-length data item to procedural operations is the same as for any other data item. A variable-length data item may be used in any procedural operation where a non-variable-length data item may be used.

The persistence of a variable-length data item is the same as that of a non-variable-length data item. Although the memory resources assigned to the data item may be physically remote from those used by neighboring non-variable-length data items, this has no effect on the results of the execution of the program.

The resources used by a variable-length data item may be freed automatically when:

1) an object instance containing the data item is finalized;

2) the program containing the data item is an INITIAL program and executes an EXIT PROGRAM or GOBACK;

3) the program containing the data item is a main program and executes a STOP RUN;

4) the program containing the data item is a called program and a CANCEL is executed referring to it;

5) the program containing the data item encounters a fatal exception;

6) the variable-length data item is explicitly or implicitly reduced in size;

7) the variable-length data item is overwritten with a new value.

The actual time when the resources used by a variable-length data item are freed is implementor-defined.

8.5.1.9 Variable-length groups

A variable-length group is a group item whose data description has at least one any-length elementary item or dynamic-capacity table as a subordinate item. All other group items are referred to as fixed-length groups.

Unlike other group items, a variable-length group is not equivalent to an alphanumeric data item and cannot undergo a comparison or a move operation, in either direction, explicitly or otherwise, unless the other operand is a compatible group. Groups are compatible if all variable-length data items correspond and match as specified

below. For the purposes of compatibility, either both operands may be variable-length groups or only one of the operands may be a variable-length group.

In determining compatibility, any subordinate data items that specify the REDEFINES clause, and all data items subordinate to those data items, are ignored.

Two fixed-length groups are always compatible, unless they are strongly typed and have different type definitions. Determination of compatibility between a variable-length group and another group is as follows:

1) For each dynamic-capacity table in either group there is a corresponding table in the other group as specified in 8.5.1.9.1, Positional correspondence.

2) For each pair of corresponding tables, these tables match, as specified in 8.5.1.9.2, Matching.

3) For each any-length elementary item in either group there is a corresponding any-length elementary item in the other group as specified in 8.5.1.9.1, Positional correspondence.

8.5.1.9.1 Positional correspondence

Two any-length elementary items correspond if they start at the same relative byte positions within their groups.

Two tables correspond if at least one of them is a dynamic-capacity table and they occupy the same relative byte positions within their groups.

8.5.1.9.2 Matching

Two any-length elementary items always match, regardless of their definitions. For purposes of determining compatibility, including the calculation of relative byte positions, all any-length elementary items are considered to be of zero length.

Two corresponding tables match when the byte length of their entries is equal and their entries are compatible.

If one of the corresponding tables is not a dynamic-capacity table, that table is treated as though it were a dynamic-capacity table whose capacity is either its fixed number of occurrences or the value of the DEPENDING operand, as applicable. For purposes of determining compatibility, the dynamic-capacity table is considered to be the same length as the corresponding table.

Dynamic-capacity tables that match each other are each considered to be the length of a single element of that table.

8.5.2 Class and category of data items and literals

Each data item and each literal has a class and a category.

Both the class and the category of a strongly-typed group item are the type-name specified in the TYPE clause in the data description of the group item. The class and the category of a group item that is not strongly typed are as follows:

- an alphanumeric group item has class and category alphanumeric
- a bit group item has class and category boolean
- a national group item has class and category national.

An alphanumeric group item is treated as though it had a usage of display.

The category of an elementary data item depends upon its description. The class of an elementary data item is related to its category, as shown in Table 3, Class and category relationships for elementary data items.

Table 3 — Class and category relationships for elementary data items

Class	Category
Alphabetic	Alphabetic
Alphanumeric	Alphanumeric Alphanumeric-edited Numeric-edited (if usage is display)
Boolean	Boolean
Index	Index
National	National National-edited Numeric-edited (if usage is national)
Numeric	Numeric
Object	Object-reference
Pointer	Data-pointer Function-pointer Program-pointer

The class and category of a literal are defined in 8.3.1.2, Literals.

Use of the name of a data class or data category in the rules of COBOL refers to the category unless class is specifically indicated.

8.5.2.1 Alphabetic category

An elementary data item described as alphabetic by its PICTURE character-string is of category alphabetic.

Such an item is referred to as an alphabetic data item.

An alphabetic data item may be specified wherever an elementary alphanumeric data item is permitted as an operand. Where an alphabetic data item is not expressly permitted, the alphabetic data item is treated as though it were an alphanumeric data item.

8.5.2.2 Alphanumeric category

Each of the following is a data item of category alphanumeric:

1) An elementary data item described as alphanumeric by its PICTURE character-string.

2) An elementary data item described with a VALUE clause containing an alphanumeric literal, and without a PICTURE clause.

3) An alphanumeric group item.

4) An alphanumeric function.

Such an item is referred to as an alphanumeric data item.

8.5.2.3 Alphanumeric-edited category

An elementary data item described as alphanumeric-edited by its PICTURE character-string is of category alphanumeric-edited.

Such an item is referred to as an alphanumeric-edited data item.

8.5.2.4 Boolean category

Each of the following is a data item of category boolean:

1) An elementary data item described as boolean by its PICTURE character-string.

2) An elementary data item described with a VALUE clause containing a boolean literal, and without a PICTURE clause.

3) A group item explicitly or implicitly described with a GROUP-USAGE clause with the BIT phrase.

4) A boolean function.

Such an item is referred to as a boolean data item.

8.5.2.5 Data-pointer category

An elementary data item explicitly or implicitly described as usage data-pointer is of category data-pointer.

Such an item is referred to as a data-pointer or as a data-pointer data item.

8.5.2.6 Function-pointer category

An elementary data item explicitly or implicitly described as usage function-pointer is of category function-pointer.

Such an item is referred to as a function-pointer or as a function-pointer data item.

8.5.2.7 Index category

Each of the following is a data item of category index:

1) An elementary data item explicitly or implicitly described as usage index.

2) An index function.

Such an item is referred to as an index data item.

8.5.2.8 National category

Each of the following is a data item of category national:

1) An elementary data item described as national by its PICTURE character-string.

2) An elementary data item described with a VALUE clause containing a national literal and described without a PICTURE clause.

3) A group item explicitly or implicitly described with a GROUP-USAGE clause with the NATIONAL phrase.

4) A national function.

Such an item is referred to as a national data item.

8.5.2.9 National-edited category

An elementary data item described as national-edited by its PICTURE character-string is of category national-edited.

Such an item is referred to as a national-edited data item.

8.5.2.10 Numeric category

Each of the following is a data item of category numeric:

1) An elementary data item described as numeric by its PICTURE character-string and not described with a BLANK WHEN ZERO clause.

2) An elementary data item described with one of the following usages: binary-char, binary-short, binary-long, binary-double, float-short, float-long, float-extended, float-binary-7, float-binary-16, float-binary-34, float-decimal-16, or float-decimal-34.

3) A LINE-COUNTER.

4) A LINAGE-COUNTER.

5) A PAGE-COUNTER.

6) A numeric function.

7) An integer function.

Such an item is referred to as a numeric data item.

8.5.2.11 Numeric-edited category

Each of the following is a data item of category numeric-edited:

1) A data item described as numeric-edited by its PICTURE character-string.

2) A data item described as numeric by its PICTURE character-string and described with a BLANK WHEN ZERO clause.

Such an item is referred to as a numeric-edited data item.

8.5.2.12 Object-reference category

An elementary data item explicitly or implicitly described as usage object-reference is of category object-reference.

Such an item is referred to as an object-reference.

8.5.2.13 Program-pointer category

An elementary data item explicitly or implicitly described as usage program-pointer is of category program-pointer.

Such an item is referred to as a program-pointer or as a program-pointer data item.

8.5.3 Types

A type is a template that contains all the characteristics of a data item and its subordinates. A type is declared and named by specifying the TYPEDEF clause. A type is referenced in a data description entry by specifying the TYPE clause. The essential characteristics of a type, which is identified by its type-name, are the relative positions and lengths of the elementary items defined in the type declaration, and the BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, SYNCHRONIZED, and USAGE clauses specified for each of these elementary items.

A type is referenced by specifying a data description entry with the TYPE clause. The typed item defined by this specification has all the characteristics of the referenced type.

Group items can be strongly or weakly typed. A typed group item is strongly typed in any of the following cases:

- The item is described with a TYPE clause that references a type declaration specifying the STRONG phrase.
- The item is subordinate to a group item described with the TYPE clause that references a type declaration specifying the STRONG phrase.

Elementary items cannot be strongly typed.

Two typed items are of the same type when:

- The items are described with TYPE clauses that reference equivalent type declarations; or
- The items are described as subordinate items in equivalent type declarations, starting at the same relative byte position and having the same length in bytes.

Two type declarations are considered equivalent when they have the same type-name, and for each elementary item in one type declaration there is a corresponding elementary item in the other type declaration, starting at the same relative byte position and having the same length in bytes. Each pair of corresponding elementary items shall have the same BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, SYNCHRONIZED, and USAGE clauses, with the following exceptions:

1) Currency symbols match if and only if the corresponding currency strings are the same.

2) Period picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these type declarations. Comma picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these type declarations.

Additionally, locale specifications in the PICTURE clauses match if and only if:

- both specify the same SIZE phrase in the LOCALE phrase of the PICTURE clause, and

- both specify the LOCALE phrase without a locale-name or both specify the LOCALE phrase with the same external identification, where the external identification is the external-locale-name or literal value associated with a locale-name in the LOCALE clause of the SPECIAL-NAMES paragraph.

NOTE When two typed items are defined in the same source element, the above rules mean that either both items are described with the same TYPE clause or both items are described as the same subordinate item in the same type declaration.

8.5.3.1 Weakly-typed items

Weakly-typed items have the characteristics of their corresponding type declarations. These characteristics cannot be overridden by specifications on a group superordinate to the typed item.

Weakly-typed items are used in the same manner as untyped items.

NOTE The type declaration can be regarded as a 'shorthand' for one or more data description entries.

8.5.3.2 Strongly-typed group items

Like weakly-typed items, strongly-typed group items have the characteristics of their corresponding type declarations. Additionally, use of a strongly-typed group item is subject to restrictions to protect the integrity of the data.

The only kind of items that may be strongly typed are group items.

8.5.4 Zero-length items

A zero-length item is a data item whose minimum length is zero and whose length at runtime is zero. A zero-length item is one of the following:

1) A group data item containing only an occurs-depending table in which the number of occurrences is zero.

2) A group data item containing only a subordinate zero-length item.

3) A data item defined with the ANY LENGTH clause that is a zero-length item.

4) A logical record that has been specified using the variable-length or the fixed-or-variable-length format of the RECORD clause in which the number of characters positions is zero.

5) An intrinsic function that returns a zero-length value.

6) A variable-length group containing only dynamic-capacity tables each of whose current capacity is zero.

8.6 Scope and life cycle of data

A source unit may contain other source units, and these contained source units may reference some of the resources of the source unit in which they are contained. (See 10, Structured compilation group, for full details of the structure.)

8.6.1 Global names and local names

The scope of global and local names is described in 8.4.5, Scope of names.

8.6.2 External and internal items

Accessible data items require that certain representations of data be stored. File connectors require that certain information concerning files be stored. The storage associated with a data item or a file connector may be external or internal.

A record described in the working-storage section is given the external attribute by the presence of the EXTERNAL clause in its data description entry. Any data item described by a data description entry subordinate to an entry describing an external record also attains the external attribute. If a record or data item does not have the external attribute, it is internal.

A file connector is given the external attribute by the presence of the EXTERNAL clause in the associated file description entry. If the file connector does not have the external attribute, it is internal.

The records described subordinate to a file description entry that does not contain the EXTERNAL clause or a sort-merge file description entry, as well as any data items described subordinate to the data description entries for such records, are always internal. If the EXTERNAL clause is included in the file description entry, its records and their subordinate data items attain the external attribute.

Records, subordinate data items, and various associated control information described in the local-storage, linkage, report, and screen sections are always internal. Special considerations apply to data described in the linkage section whereby an association is made between the records described and other data items accessible to other runtime elements.

External and internal data items and file connectors may have either global or local names.

If a data item or file connector is external, the storage associated with that item is associated with the run unit rather than with any particular runtime element within the run unit. An external item may be accessed by any runtime element in the run unit that describes it. References to external items from different runtime elements using separate descriptions of the data item or file connector are always references to the same item. In a run unit, there is only one representation of an external item.

If a data item or file connector is internal, the storage associated with it is associated only with the runtime module that describes it. Internal items are described in 8.6.3, Automatic, initial, and static internal items.

8.6.3 Automatic, initial, and static internal items

Each internal item has one of the three persistence attributes: automatic, initial, or static. The designation of automatic, initial, and static items relates to their persistence and the persistence of their contents during the execution of the run unit.

Data items, file connectors, and screen item attributes have two states: initial and last-used. The initial state of a data item depends on the presence or absence of a VALUE clause in its data description entry, the section in which the data item is described, and the description of the data item. The initial state of a file connector is that it is not in an open mode. The initial state of a screen item attribute depends on the description of the screen item.

Last-used state means that the content of the data item, file connector, or screen item attribute is that of the last time it was modified.

Data items defined in the local-storage section are automatic items. Their storage is allocated and set to initial state each time the runtime element containing them is activated. Each activated instance of the runtime element has its own copy of the item that persists while that instance of the runtime element is in active state.

Data items and file connectors defined in the file and working-storage sections of an initial program are initial items. Also, screen item attributes in an initial program are treated as initial items. They are set to their initial state each time an initial program is activated. An initial item persists while the program is in active state. It is undefined whether each activation of an initial program has its own copy of initial items.

Data items and file connectors defined in the working-storage or file section of a source element that is not an initial program are static items. Also, screen item attributes in a source element that is not an initial program are treated as static items. These items are set to their initial state each time the runtime element or object containing them is set to its initial state, as described in 14.6.2.1.2.1, Initial state, and in 14.6.2.2, Initial state of object data. They are allocated no later than immediately before initialization and persist to the first of the following:

— the end of the run unit,
— the execution of a CANCEL statement of a program that directly or indirectly contains the items,
— the end of the object's life cycle in the case of object data.

For static items that are not object data, there is one copy in a run unit.

For static items that are object data:

— In the factory object of a given class, there is one copy of each static item that is described in or inherited by the factory definition of that class.

— In each instance object of a given class, there is one copy of each static item that is described in or inherited by the instance definition of that class.

Further details are specified in 9.3.1, Objects and classes, 9.3.9, Class inheritance, and 9.3.14, Object life cycle.

Data items described in the linkage section, when specified as formal parameters or the returning item in an activated source element, have the same persistence attributes as their corresponding arguments or the corresponding returning item in the activating source element. They are in last-used state when the runtime element is activated, and they persist while the argument persists and the source element that contains their definition is in active state.

A table index is treated as a static item if the associated table is static and as an automatic item if the associated table is automatic.

The persistence of a variable-length data item is the same as for any other data item defined in the same data division section. A variable-length data item is in its initial state at the start of processing and reverts to its initial state under the same circumstances as defined for other data items. A dynamic-capacity table in its initial state has its initial capacity set as specified in 8.5.1.6.3, Dynamic-capacity tables. When the data description entry of an

any-length elementary item contains a VALUE clause, the rules of the VALUE clause define the length of that item in its initial state. If no VALUE clause is specified, the length of that item in its initial state is zero.

A structured constant is a static item that is always in initial state.

8.6.4 Based entries and based data items

A based entry is a data description entry in the working-storage section, local-storage section, or linkage section that is described with a BASED clause. A based entry is not initially associated with an actual data item. An association is established linking the based entry to actual data when its implicit data-address pointer is assigned the address of an existing data item or assigned the address of storage obtained with an ALLOCATE statement. This association establishes a based data item. The association is maintained in an implicit data-address pointer that may be referenced by an identifier of the form ADDRESS OF data-name, where data-name is the name of the based entry.

The association may cease to exist because the actual data no longer exists, as specified 8.6.3, Automatic, initial, and static internal items, and in 14.9.3, ALLOCATE statement, or because the implicit data-address pointer no longer references the actual data.

The association ends:

— when its implicit data-address pointer is set to a different value,

— when the based entry is defined in the working-storage or local-storage section, at the end of the life cycle of the data items defined in that section,

— when the based entry is defined in the linkage section, at the end of the execution of the runtime element.

8.6.5 Common, initial, and recursive attributes

A program can be described with attributes that affect its initial state or that define the manner in which it may be called.

A common program is one that is directly contained within another program and that can be called by programs directly or indirectly contained in that other program, as described in 8.4.5.2, Scope of program-names. The common attribute is attained by specifying the COMMON clause in a program's identification division. When the COMMON clause is not specified, a contained program that is not recursive may be called only from the directly-containing program. The COMMON clause facilitates the writing of nested programs that can be used by all the programs contained within a program.

An initial program is one whose program state is initialized when the program is called. During the process of initializing an initial program, that program's internal data is initialized as described in 14.6.2, State of a function, method, object, or program. The initial attribute is attained by specifying the INITIAL clause in the program's identification division.

A recursive program may call itself directly or indirectly. The program's internal data is initialized as described in 14.6.2, State of a function, method, object, or program. The recursive attribute is attained by specifying the RECURSIVE clause in the program’s identification division.

Functions and methods are always recursive. Their data is initialized in the same way as recursive programs.

If neither the INITIAL nor RECURSIVE clause is specified in a program's identification division, the program's data is in the last-used state on other than the first activation of the program as described in 14.6.2, State of a function, method, object, or program. The program cannot be activated while it is active unless RECURSIVE is specified.

8.6.6 Sharing data items

Two runtime elements in a run unit may reference common data in the following circumstances:

1) The data content of an external data record may be referenced from any runtime element provided that the runtime element has described that record.

2) If a program is contained within another program, both programs may refer to data possessing the global attribute either in the containing program or in any program that directly or indirectly contains the containing program.

3) The mechanism whereby an argument value is passed by reference from an activating runtime element to an activated runtime element establishes a common data item. The activated unit and the activating unit may use a different name to refer to the common data item.

8.7 Operators

8.7.1 Arithmetic operators

There are five binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific COBOL characters that shall be preceded by a space and followed by a space except that no space is required between a left parenthesis and a unary operator or between a unary operator and a left parenthesis. The following are the arithmetic operators:

Binary Arithmetic Operators	**Meaning**
+	Addition
–	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Arithmetic Operators	**Meaning**
+	The effect of multiplication by the numeric literal +1
–	The effect of multiplication by the numeric literal –1

8.7.2 Boolean operators

A boolean operator specifies the type of boolean operation to be performed on one or two operands, for a unary operator or binary operator, respectively. The following are the boolean operators:

Binary boolean operators	**Meaning**
B-AND	AND operation (boolean conjunction)
B-OR	Inclusive OR operation (boolean inclusive disjunction)
B-XOR	Exclusive OR operation (boolean exclusive disjunction)

Unary boolean operator	**Meaning**
B-NOT	Negation operation

8.7.3 Concatenation operator

The concatenation operator is the COBOL character '&', which shall be immediately preceded and followed by a separator space.

8.7.4 Invocation operator

The invocation operator is the two contiguous COBOL characters '::', which shall be immediately preceded and followed by a separator space. The use of the invocation operator is given in 8.4.2.4, Inline method invocation.

8.7.5 Relational operators

The relational operators specify the type of comparison to be made in a relation condition.

8.7.5.1 General format

Format 1 (simple-relational-operator):

```
{ IS GREATER THAN  }
{ IS >             }
{ IS LESS THAN     }
{ IS <             }
{ IS EQUAL TO      }
{ IS =             }
```

Format 2 (extended-relational-operator):

```
{ IS GREATER THAN OR EQUAL TO }
{ IS >=                       }
{ IS NOT LESS THAN            }
{ IS NOT <                    }
{ IS LESS THAN OR EQUAL TO    }
{ IS <=                       }
{ IS NOT GREATER THAN         }
{ IS NOT >                    }
{ IS NOT EQUAL TO             }
{ IS NOT =                    }
{ IS <>                       }
```

8.7.5.2 Syntax rules

1) Format 1 specifies simple relational operators.

2) Format 2 specifies extended relational operators.

3) > is an abbreviation for GREATER THAN.

4) NOT > is an abbreviation for NOT GREATER THAN.

5) < is an abbreviation for LESS THAN.

6) NOT < is an abbreviation for NOT LESS THAN.

7) = is an abbreviation for EQUAL TO.

8) NOT = is an abbreviation for NOT EQUAL.

9) >= is an abbreviation for GREATER THAN OR EQUAL TO.

10) <= is an abbreviation for LESS THAN OR EQUAL TO.

11) <> is an abbreviation for NOT EQUAL.

8.7.6 Logical operators

The logical operators are the words AND, NOT, and OR. The use of the logical operators is given in 8.8.4.2, Complex conditions.

8.8 Expressions

8.8.1 Arithmetic expressions

An arithmetic expression may be an identifier referencing a numeric data item, a numeric literal, the figurative constant ZERO (ZEROS, ZEROES), such identifiers, figurative constants, and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of identifiers, numeric literals, arithmetic operators, and parentheses are given in table 4, Combinations of symbols in arithmetic expressions.

Evaluation rules for arithmetic expressions depend on the mode of arithmetic in effect.

8.8.1.1 Native, standard, standard-binary, and standard-decimal arithmetic

The following rules apply regardless of the mode of arithmetic that is in effect:

1) Parentheses may be used in arithmetic expressions to specify the order of the arithmetic operations. Operations within parentheses are executed before operations on the parenthesized expression; the result is treated as a single operand in any further operation. When nested parentheses are specified, operations are executed from within the least inclusive set of parentheses to the most inclusive set.

2) When operands are at the same level of inclusiveness, the following hierarchical order of execution is implied:

   1st — Unary plus and minus
   2nd — Exponentiation
   3rd — Multiplication and division
   4th — Addition and subtraction

3) When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

   NOTE Parentheses are used to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence, or to emphasize the normal sequence for the sake of clarity.

4) The ways in which identifiers, literals, operators, and parentheses may be combined in arithmetic expressions are summarized in Table 4, Combinations of symbols in arithmetic expressions.

Table 4 — Combinations of symbols in arithmetic expressions

First symbol	Second symbol				
	Identifier or literal	**+ – * / ****	**Unary + or –**	**(**	**)**
Identifier or literal	—	P	—	—	P
+ – * / **	P	—	P	P	—
Unary **+** or **–**	P	—	—	P	—
(	P	—	P	P	—
)	—	P	—	—	P
The letter 'P' indicates a permissible pair of symbols. The character '—' indicates an invalid pair.					

5) An arithmetic expression may begin only with the symbol '(', '+', '–', an identifier, or a literal and may end only with a ')', an identifier, or a literal. There shall be a one-to-one correspondence between left and right

parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis. If the first operator in an arithmetic expression is a unary operator, it shall be immediately preceded by a left parenthesis if that arithmetic expression immediately follows an identifier or another arithmetic expression.

NOTE For example, when '1' and '+ 2' are used as subscripts for a two-dimensional table A, the arithmetic expression '+ 2' needs to be enclosed in parentheses, as in A (1 (+ 2)).

6) The following rules apply to evaluation of exponentiation in an arithmetic expression:

   a) If the value of an expression to be raised to a power is zero, the exponent shall have a value greater than zero. Otherwise, the EC-SIZE-EXPONENTIATION exception condition is set to exist and the size error condition is raised.

   b) If the evaluation yields both a positive and a negative real number, the value returned as the result is the positive number.

   c) If the value of an expression to be raised to a power is less than zero, the evaluation of the exponent shall result in an integer. Otherwise, the EC-SIZE-EXPONENTIATION exception condition is set to exist and the size error condition is raised.

7) Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and receiving data items.

8.8.1.2 Native arithmetic

Native arithmetic is an implementor-defined method of evaluating an arithmetic expression, an arithmetic statement, the SUM clause, and all integer and numeric functions. Native arithmetic is in effect when the ARITHMETIC IS NATIVE clause is specified in the OPTIONS paragraph or no ARITHMETIC clause is specified. It will also be in effect when the ARITHMETIC IS STANDARD clause is specified and certain operations use data items as indicated in 8.8.1.3, Standard arithmetic. The implementor shall specify techniques used for native arithmetic.

8.8.1.3 Standard arithmetic

Standard arithmetic is a standard method of evaluating an arithmetic expression, an arithmetic statement, the SUM clause, and certain integer and numeric functions as specified in 15.4.1, Numeric and integer functions. The rules for standard arithmetic are defined in 8.8.1.3.1, Standard intermediate data item through 8.8.1.3.8, Unary minus. Standard arithmetic is in effect when the ARITHMETIC IS STANDARD clause is specified in the OPTIONS paragraph. When a data item described with a usage of binary-char, binary-short, binary-long, binary-double, float-short, float-long, or float-extended is an operand in an arithmetic expression, an arithmetic statement, the SUM clause, or certain integer and numeric functions, the techniques used shall be those specified for native arithmetic.

When standard arithmetic has been specified or implied for the source unit and the FLAG-NATIVE-ARITHMETIC directive is enabled, any arithmetic operation that uses native arithmetic shall be flagged.

NOTE Standard arithmetic is an obsolete feature of standard COBOL.

8.8.1.3.1 Standard intermediate data item

A standard intermediate data item is of the class numeric and the category numeric. It is the unique value zero or an abstract, signed, normalized decimal floating-point temporary data item. The internal representation shall be defined by the implementor. Implementors may use whatever method or methods they wish as long as the results conform to the rules for standard intermediate data items.

NOTE The internal representation of a standard intermediate data item is permitted to vary so that implementors are able to choose the most efficient implementation for the circumstances.

When standard arithmetic is in effect the following rules apply:

1) Any operand of an arithmetic expression that is not already contained in a standard intermediate data item shall be converted into a standard intermediate data item.

   NOTE This rule covers such cases as an arithmetic expression that contains only one operand and no operator. For example, IF (A = 1) and COMPUTE A = B.

2) The size error condition is raised and the EC-SIZE-OVERFLOW or EC-SIZE-UNDERFLOW exception condition is set to exist if the value is too large or too small, respectively, to be contained in a standard intermediate data item. (See 14.7.4, SIZE ERROR phrase and size error condition.)

   NOTE Underflow is treated as a SIZE ERROR and is not rounded to zero.

8.8.1.3.1.1 Precision and allowable magnitude

A standard intermediate data item has the unique value of zero or a value whose magnitude is in the range of

10^{-999} (1.000 000 000 000 000 000 000 000 000 000 0E-999)

through

$10^{1000} - 10^{968}$ (9.999 999 999 999 999 999 999 999 999 999 9E+999)

inclusive, with a precision of 32 decimal digits. A standard intermediate data item shall be truncated or rounded to fewer than 32 digits only as explicitly specified.

8.8.1.3.1.2 Normalized values

When the value of a standard intermediate data item is not zero, the significand shall contain exactly one nonzero digit to the left of the decimal point.

8.8.1.3.1.3 Rounding rules for standard arithmetic

The ability to specify the rounding rules that apply to standard intermediate data items is described in 11.9.7, INTERMEDIATE ROUNDING clause.

8.8.1.3.1.3.1 Rounding rules for INTERMEDIATE ROUNDING IS NEAREST-AWAY-FROM-ZERO

If INTERMEDIATE ROUNDING IS NEAREST-AWAY-FROM-ZERO is specified or implied, a standard intermediate data item shall be rounded to 31 digits in the situations listed below, and the rounding shall occur as described for the NEAREST-AWAY-FROM-ZERO clause of the ROUNDED phrase.

1) When a standard intermediate data item is compared except when the comparison is among the arguments of an intrinsic function, in which case it shall not be rounded and all 32 digits shall be used.

2) When a standard intermediate data item is the argument of a function and there is no equivalent arithmetic expression defined for the rules of the function, unless otherwise specified in the rules for a function or unless situation 1, above, applies.

3) When a standard intermediate data item is being moved to a resultant-identifier for which the ROUNDED phrase has not been specified. Rounding of a standard intermediate data item may cause the size error condition to be raised or the EC-SIZE-OVERFLOW exception condition to exist.

When a standard intermediate data item is being moved to a resultant-identifier for which the ROUNDED phrase is specified, the number of digits to which rounding occurs is as specified in 14.7.3, ROUNDED phrase.

8.8.1.3.1.3.2 Rounding rules for INTERMEDIATE ROUNDING IS NEAREST-EVEN

If INTERMEDIATE ROUNDING IS NEAREST-EVEN is specified, a standard intermediate data item shall be rounded to 31 digits in the situations listed below, and the rounding shall occur as described for the NEAREST-EVEN clause of the ROUNDED phrase.

1) When a standard intermediate data item is compared except when the comparison is among the arguments of an intrinsic function, in which case it shall not be rounded and all 32 digits shall be used.

2) When a standard intermediate data item is the argument of a function and there is no equivalent arithmetic expression defined for the rules of the function, unless otherwise specified in the rules for a function or unless situation 1, above, applies.

3) When a standard intermediate data item is being moved to a resultant-identifier for which the ROUNDED phrase has not been specified. Rounding of a standard intermediate data item may cause the size error condition to be raised or the EC-SIZE-OVERFLOW exception condition to exist.

When a standard intermediate data item is being moved to a resultant-identifier for which the ROUNDED phrase is specified, the number of digits to which rounding occurs is as specified in 14.7.3, ROUNDED phrase.

8.8.1.3.1.3.3 Rounding rules for INTERMEDIATE ROUNDING IS PROHIBITED

If INTERMEDIATE ROUNDING IS PROHIBITED is specified, and the intermediate value cannot be represented exactly in a standard intermediate data item, the EC-SIZE-TRUNCATION exception is set to exist and the results of the operation are undefined.

8.8.1.3.1.3.4 Rounding rules for INTERMEDIATE ROUNDING IS TRUNCATION

If INTERMEDIATE ROUNDING IS TRUNCATION is specified or implied, a standard intermediate data item shall be rounded to 31 digits in the situations listed below and the rounding shall occur as described for the ROUNDED phrase.

1) When a standard intermediate data item is compared except when the comparison is among the arguments of an intrinsic function, in which case it shall not be rounded and all 32 digits shall be used.

2) When a standard intermediate data item is the argument of a function and there is no equivalent arithmetic expression defined for the rules of the function, unless otherwise specified in the rules for a function or unless situation 1, above, applies.

3) When a standard intermediate data item is being moved to a resultant-identifier for which the ROUNDED phrase has not been specified. Rounding of a standard intermediate data item may cause the size error condition to be raised or the EC-SIZE-OVERFLOW exception condition to exist.

When a standard intermediate data item is being moved to a resultant-identifier for which the ROUNDED phrase is specified, the number of digits to which rounding occurs is as specified in 14.7.3, ROUNDED phrase.

8.8.1.3.2 Addition

For addition, the operands and operator are: operand-1 + operand-2. The result shall be the exact sum truncated to 32 significant digits, normalized, and stored in a standard intermediate data item.

8.8.1.3.3 Subtraction

For subtraction, the operands and operator are: operand-1 – operand-2. The result shall be equivalent to the evaluation of the arithmetic expression

(operand-1 + (– operand-2))

8.8.1.3.4 Multiplication

For multiplication, the operands and operator are: operand-1 * operand-2. The result shall be the exact product truncated to 32 significant digits, normalized, and stored in a standard intermediate data item.

8.8.1.3.5 Division

For division, the operands and operator are: operand-1 / operand-2. The result shall be the exact quotient truncated to 32 significant digits, normalized, and stored in a standard intermediate data item.

8.8.1.3.6 Exponentiation

For exponentiation, the operands and operator are: operand-1 ** operand-2.

1) When operand-2 is zero and operand-1 is other than zero, the result shall be equivalent to the evaluation of the arithmetic expression

   (1)

2) When the value of operand-2 is greater than zero, the results shall be determined as follows:

   a) When operand-2 is an integer and the following condition is true

      (operand-2 = 1)

      the equivalent expression shall be

      (operand-1)

   b) When operand-2 is an integer and the following condition is true

      (operand-2 = 2)

      the equivalent expression shall be

      (operand-1 * operand-1)

   c) When operand-2 is an integer and the following condition is true

      (operand-2 = 3)

      the equivalent expression shall be

      ((operand-1 * operand-1) * operand-1)

   d) When operand-2 is an integer and the following condition is true

      (operand-2 = 4)

      the equivalent expression shall be

      ((operand-1 * operand-1) * (operand-1 * operand-1))

   e) Otherwise, the value of the result shall be an implementor-defined value that is normalized and stored in a standard intermediate data item.

3) When operand-2 is less than zero, the result shall be equivalent to the evaluation of the arithmetic expression

(1 / (operand-1 ** FUNCTION ABS (operand-2)))

4) When both operand-1 and operand-2 are equal to zero, the EC-SIZE-EXPONENTIATION exception condition is set to exist.

8.8.1.3.7 Unary plus

For unary plus, the operator and operand are: + operand. The result shall be equivalent to the evaluation of the arithmetic expression

(operand)

8.8.1.3.8 Unary minus

For unary minus, the operator and operand are: – operand. The result shall be equivalent to the evaluation of the arithmetic expression

(–1 * operand)

8.8.1.4 Standard-binary arithmetic

Standard-binary arithmetic is a method of evaluating an arithmetic expression, an arithmetic statement, the SUM clause, and certain integer and numeric functions as specified in 15.4.1, Numeric and integer functions, in a manner consistent with the rules associated with binary floating-point arithmetic as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic is in effect when the STANDARD-BINARY phrase of the ARITHMETIC clause is specified in the OPTIONS paragraph.

The mechanisms for conversion of data items described with a usage of float-short, float-long, or float-extended into standard binary intermediate data items are defined by the implementor.

The EC-DATA-INCOMPATIBLE exception condition is set to exist when an operand, or a combination of operands, results in the signaling of an 'invalid operation' state as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, section 7.1, Invalid Operation.

8.8.1.4.1 Standard-binary intermediate data item

A standard-binary intermediate data item is of the class numeric and the category numeric. The format of a standard-binary intermediate data item is the same as for a data item described with the FLOAT-BINARY-34 phrase of the USAGE clause. This format is defined in IEEE STD 754-2008, IEEE Standard for Floating-Point Arithmetic, as the 'Basic 128-bit Binary Format' and is hereinafter described as 'binary128 format' for convenience.

When standard-binary arithmetic is in effect, the following rules apply:

1) Any operand of an arithmetic expression that is not already in binary128 format is converted into this form.

2) The size error condition is set to exist and the EC-SIZE-OVERFLOW or EC-SIZE-UNDERFLOW exception condition is set to exist if the value is too large or too small, respectively, to be contained in an item in binary128 format as specified in 14.7.4, SIZE ERROR phrase and size error condition.

3) The rounding rules that apply to standard-binary intermediate data items are described in 11.9.7, INTERMEDIATE ROUNDING clause.

4) The behavior of exponentiation is described in 8.8.1.4.3, Exponentiation in standard-binary arithmetic.

8.8.1.4.2 Basic arithmetic operations in standard-binary arithmetic

Addition, subtraction, multiplication, and division are performed as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 5.4.1, Arithmetic operations. The operations used are the general computation

operations formatOf-addition, formatOf-subtraction, formatOf multiplication, and formatOf-division. Unary plus and unary minus are performed as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 5.5.1, Sign bit operations.

8.8.1.4.3 Exponentiation in standard-binary arithmetic

For exponentiation, the operands and operator are: operand-1 ** operand-2.

1) When operand-2 is zero and operand-1 is other than zero, the result shall be equivalent to the evaluation of the arithmetic expression

   (1)

2) When the value of operand-2 is greater than zero, the results shall be determined as follows:

   a) When operand-2 is an integer and the following condition is true

      (operand-2 = 1)

      the equivalent expression shall be

      (operand-1)

   b) When operand-2 is an integer and the following condition is true

      (operand-2 = 2)

      the equivalent expression shall be

      (operand-1 * operand-1)

   c) When operand-2 is an integer and the following condition is true

      (operand-2 = 3)

      the equivalent expression shall be

      ((operand-1 * operand-1) * operand-1)

   d) When operand-2 is an integer and the following condition is true

      (operand-2 = 4)

      the equivalent expression shall be

      ((operand-1 * operand-1) * (operand-1 * operand-1))

   e) Otherwise, the result shall be an implementor-defined value. Operands used in the development of that value shall be standard-binary intermediate data items. All additions, subtractions, multiplications and divisions performed in the development of the result shall be performed in accordance with the corresponding rules in IEEE P754 for basic 128-bit binary floating-point operands.

3) When operand-2 is less than zero, the result shall be equivalent to the evaluation of the arithmetic expression

   (1 / (operand-1 ** FUNCTION ABS (operand-2)))

4) When both operand-1 and operand-2 are equal to zero, the EC-SIZE-EXPONENTIATION exception condition is set to exist.

8.8.1.5 Standard-decimal arithmetic

Standard-decimal arithmetic is a method of evaluating an arithmetic expression, an arithmetic statement, the SUM clause, and certain integer and numeric functions as specified in 15.4.1, Numeric and integer functions, in a manner consistent with the rules associated with decimal floating-point arithmetic as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic. Standard-decimal arithmetic is in effect when the STANDARD-DECIMAL phrase of the ARITHMETIC clause is specified in the OPTIONS paragraph.

The mechanisms for conversion of data items described with a usage of float-short, float-long, or float-extended into standard-decimal intermediate data items are defined by the implementor.

The EC-DATA-INCOMPATIBLE exception condition is set to exist when an operand, or a combination of operands, results in the signaling of an 'invalid operation' state as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, section 7.1, Invalid Operation.

8.8.1.5.1 Standard-decimal intermediate data item

A standard-decimal intermediate data item is of the class numeric and the category numeric. The format of a standard-decimal intermediate data item is the same as for a data item described with the FLOAT-DECIMAL-34 phrase of the USAGE clause. This format is defined in IEEE STD 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.5.2, Encodings, as the 'Basic 128-bit Decimal Format' using the decimal encoding. This format is hereinafter described as decimal128 format' for convenience.

If the content of the standard-decimal intermediate data item is a finite numeric value, the content is in canonical form as described in IEEE Std 754-2008, IEEE Standard for floating-point arithmetic, 3.5.2, Encodings, for the 128-bit decimal interchange format using decimal encoding.

When standard-decimal arithmetic is in effect, the following rules apply:

1) Any operand of an arithmetic expression that is not already in decimal128 format is converted into this form.

2) The size error condition is set to exist and the EC-SIZE-OVERFLOW or EC-SIZE-UNDERFLOW exception condition is set to exist if the value is too large or too small, respectively, to be contained in an item in decimal128 format. Refer to 14.7.4, SIZE ERROR phrase and size error condition, for more information).

3) The rounding rules that apply to standard-decimal intermediate data items are described in 11.9.7, INTERMEDIATE ROUNDING clause.

4) The behavior of exponentiation is described in 8.8.1.5.3, Exponentiation in standard-decimal arithmetic.

8.8.1.5.2 Basic arithmetic operations in standard-decimal arithmetic

Addition, subtraction, multiplication, and division are performed as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 5.4.1, Arithmetic operations. The operations used are the general computation operations formatOf-addition, formatOf-subtraction, formatOf multiplication, and formatOf-division. Unary plus and unary minus are performed as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 5.5.1, Sign bit operations.

8.8.1.5.3 Exponentiation in standard-decimal arithmetic

For exponentiation, the operands and operator are: operand-1 ** operand-2.

1) When operand-2 is zero and operand-1 is other than zero, the result shall be equivalent to the evaluation of the arithmetic expression

   (1)

2) When the value of operand-2 is greater than zero, the results shall be determined as follows:

a) When operand-2 is an integer and the following condition is true

(operand-2 = 1)

the equivalent expression shall be

(operand-1)

b) When operand-2 is an integer and the following condition is true

(operand-2 = 2)

the equivalent expression shall be

(operand-1 * operand-1)

c) When operand-2 is an integer and the following condition is true

(operand-2 = 3)

the equivalent expression shall be

((operand-1 * operand-1) * operand-1)

d) When operand-2 is an integer and the following condition is true

(operand-2 = 4)

the equivalent expression shall be

((operand-1 * operand-1) * (operand-1 * operand-1))

e) Otherwise, the result shall be an implementor-defined value. Operands used in the development of that value shall be standard-decimal intermediate data items. All additions, subtractions, multiplications and divisions performed in the development of the result shall be performed in accordance with the corresponding rules in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, for basic 128-bit decimal floating-point operands, using the decimal encoding.

3) When operand-2 is less than zero, the result shall be equivalent to the evaluation of the arithmetic expression

(1 / (operand-1 ** FUNCTION ABS (operand-2)))

4) When both operand-1 and operand-2 are equal to zero, the EC-SIZE-EXPONENTIATION exception condition is set to exist.

8.8.2 Boolean expressions

A boolean expression may be:

— an identifier referencing a boolean data item,
— a boolean literal,
— the figurative constant ZERO (ZEROS, ZEROES),
— the figurative constant ALL literal, where literal is a boolean literal,
— a boolean expression preceded by a unary boolean operator,
— two boolean expressions separated by a binary boolean operator, or
— a boolean expression enclosed in parentheses.

The following are formation and evaluation rules for boolean expressions:

1) A boolean expression shall begin with one of the following:

   — the symbol '('
   — an identifier that references a boolean data item
   — a boolean literal
   — the unary operator B-NOT.

2) A boolean expression shall end with one of the following:

   — the symbol ')'
   — an identifier that references a boolean data item
   — a boolean literal.

3) There shall be a one-to-one correspondence between left and right parentheses such that each left parenthesis shall be to the left of its corresponding right parenthesis.

4) The two operands in a binary boolean operation shall not both be the figurative constant ALL literal.

5) The permissible combinations of operands, operators, and parentheses in a boolean expression are specified in table 5, Combination of symbols in boolean expressions.

Table 5 — Combination of symbols in boolean expressions

First Symbol	Second symbol				
	Identifier or literal	**B-AND B-OR B-XOR**	**B-NOT**	**(**	**)**
Identifier or literal	—	P	—	—	P
B-AND, B-OR, B-XOR	P	—	P	P	—
B-NOT	P	—	—	P	—
(	P	—	P	P	—
)	—	P	—	—	P
Legend: P indicates a permissible pair — indicates an invalid pair					

6) Evaluation of a boolean expression shall proceed as follows:

   a) Expressions within parentheses shall be evaluated before the parenthesized expression is used in the evaluation of a more inclusive expression. Within parentheses, evaluation shall proceed from the least inclusive set of nested parentheses to the most inclusive set.

   b) The precedence of operations at the same level of inclusiveness, is:

      1st — negation (B-NOT)
      2nd — conjunction (B-AND)
      3rd — exclusive disjunction (B-XOR)
      4th — inclusive disjunction (B-OR)

   c) When the sequence of evaluation is not specified by parentheses, the evaluation of operations with the same precedence shall proceed from left to right.

      NOTE Parentheses can be used to clarify the logic where consecutive operations of the same precedence are specified or to modify the precedence when it is necessary to deviate from the normal precedence.

7) Binary boolean operations shall be performed without regard for the usage of the operands. If the two operands are of equal length, the specified operation, conjoining or disjoining (inclusively or exclusively), shall proceed by operation on boolean values in corresponding boolean positions starting from the leftmost boolean position and continuing to the rightmost boolean position. If the operands are of unequal length, the operation shall proceed as though the shorter operand were extended on the right by a sufficient number of boolean zeros to make the operands of equal length.

   NOTE Lengths are established for each boolean operation, including unary operations, in the order in which the operations are evaluated. If the operands of a Boolean operation are of zero-length, the result will be of zero-length.

8) The result of the evaluation of each boolean operation shall be a boolean value whose length shall be the number of boolean positions of the larger item referenced in that operation.

8.8.3 Concatenation expressions

A concatenation expression consists of two operands separated by the concatenation operator.

8.8.3.1 General format

{ literal-1 | concatenation-expression-1 } & literal-2

8.8.3.2 Syntax rules

1) Both operands shall be of the same class, either alphanumeric, boolean, or national, except that a figurative constant may be specified as one or both operands. Neither literal-1 nor literal-2 shall be a figurative constant that begins with the word ALL.

2) For operands of class alphanumeric, the length of the value resulting from concatenation shall be less than or equal to 8,191 alphanumeric character positions.

3) For operands of class boolean, the length of the value resulting from concatenation shall be less than or equal to 8,191 boolean character positions.

4) For operands of class national, the length of the value resulting from concatenation shall be less than or equal to 8,191 national character positions.

8.8.3.3 General rules

1) The class of the concatenation expression resulting from the concatenation operation shall be:

   a) when one of the operands is a figurative constant, the class of the literal or concatenation expression that constitutes the other operand, or

   b) when both of the operands are figurative constants, the class alphanumeric, or

   c) the same class as the operands.

2) The value of a concatenation expression shall be the concatenation of the value of the literals, figurative constants, and concatenation expressions of which it is composed. If both literal-1 and literal-2 are specified and both are zero-length literals the value of the concatenation expression is a zero-length literal.

3) A concatenation expression shall be equivalent to a literal of the same class and value, and may be used anywhere a literal of that class may be used.

8.8.4 Conditional expressions

Conditional expressions identify conditions that are tested to enable selecting one of multiple processing alternatives depending upon the truth value of the condition. A conditional expression has a truth value represented by either true or false. There are two categories of conditions associated with conditional expressions: simple conditions and complex conditions. Each may be enclosed within any number of paired parentheses, in which case its category is not changed.

8.8.4.1 Simple conditions

The simple conditions are the relation, boolean, class, condition-name, switch-status, sign, and omitted-argument conditions. A simple condition has a truth value of true or false. The inclusion in parentheses of simple conditions does not change the simple condition truth value.

8.8.4.1.1 Relation conditions

A relation condition specifies a comparison of two operands. The relational operator that joins the two operands specifies the type of comparison. A relation condition shall have a truth value of 'true' if the specified relation exists between the two operands, and a truth value of 'false' if the relation condition does not exist.

A relation condition involving operands of class boolean is a boolean relation condition; a relation condition involving operands of class pointer or object is a pointer-or-object-reference relation condition; otherwise, the relation condition is a general-relation condition.

Comparisons are defined for the following:

1) Two operands of class numeric.
2) Two operands of class alphabetic.
3) Two operands of class alphanumeric.
4) Two operands of class boolean.
5) Two operands of class national.
6) Two operands where one is a numeric integer and the other is class alphanumeric or national.
7) Two operands of different classes where each operand is from the set of classes alphanumeric, alphabetic, or national.
8) Comparisons involving indexes or index data items.
9) Two operands of class object.
10) Two operands of class pointer where each operand is of the same category.
11) Two strongly-typed operands of the same type.
12) Two compatible variable-length groups.

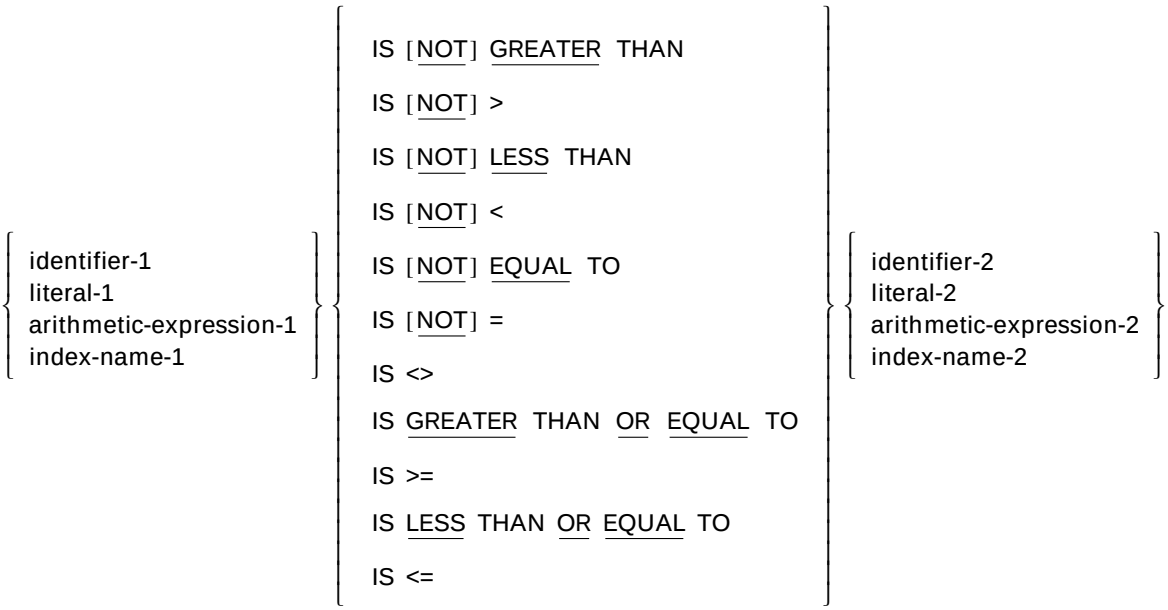
For purposes of comparison, an alphanumeric group item shall be treated as an elementary alphanumeric data item. A class alphabetic operand shall be treated as though it were an operand of class alphanumeric. A national group item or a bit group item shall be treated as an elementary national data item or an elementary bit data item, respectively.

Comparison of a variable-length group with a compatible group is defined below in 8.8.4.1.1.13, Comparison of a variable-length group with a compatible group.

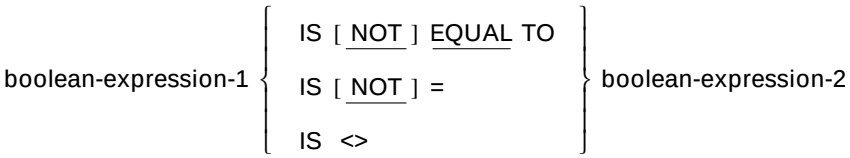
The first operand is called the subject of the condition; the second operand is called the object of the condition. A relation condition shall contain at least one reference to an operand that is not a literal.

8.8.4.1.1.1 General format

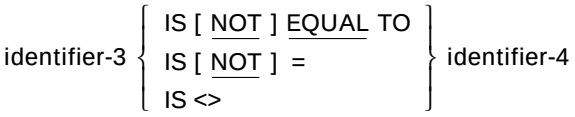
Format 1 (general-relation):



Format 2 (boolean):



Format 3 (pointer-or-object-reference):



8.8.4.1.1.2 Syntax rules

FORMAT 1

1) If either identifier-1 or identifier-2 is a strongly-typed group, both operands shall be of the same type.

2) All identifiers shall be of class alphabetic, alphanumeric, national, or numeric or shall be strongly-typed group items.

3) All literals shall be of class alphanumeric, national, or numeric.

4) Strongly-typed group items that contain elementary items of class boolean, pointer, or object-reference may be compared only for equality or inequality.

FORMAT 3

5) Identifier-3 and identifier-4 shall reference data items of class pointer or object, and both shall be of the same category.

8.8.4.1.1.3 Comparison of numeric operands

For operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands regardless of the manner in which their usage is described. The length of the literal or arithmetic expression operands, in terms of the number of digits represented, is not significant. Zero is considered a unique value regardless of the sign; an operand having the value zero is equal to any other operand having the value zero – whether positive, negative, or unsigned. When standard or native arithmetic is in effect, and one or both of the operands is an operand for which native arithmetic applies, the comparison proceeds by the rules of native arithmetic regardless of the mode of arithmetic in effect. When standard-decimal or standard-binary arithmetic is used, the number of digits of the standard-decimal or standard-binary intermediate data item, and the form of rounding used in the operation, shall be as specified in 11.9.7, INTERMEDIATE ROUNDING clause. When standard-binary arithmetic is in effect, the comparison is performed as if each operand not already in the format of a standard-binary intermediate data item had been converted to that form, and the comparison made between the two corresponding standard-binary intermediate data items. When standard-decimal arithmetic is in effect, the comparison is performed as if each operand not already in the format of a standard-decimal intermediate data item had been converted to that form, and the comparison made between the two corresponding standard-decimal intermediate data items. When standard arithmetic is used, the number of digits of the standard intermediate data item used and the form of rounding used shall be as specified in 8.8.1.3.1.3, Rounding rules for standard arithmetic.

8.8.4.1.1.4 Comparison of a numeric integer operand with an operand of class alphanumeric or national

The numeric integer operand shall be an integer literal or an integer numeric data item of usage display or national. The other operand may be a literal or data item of class alphanumeric or national.

The integer is treated as though it were moved to an elementary data item of a length the same as the number of digits in the integer and of the same class and usage as the comparand, according to the rules of the MOVE statement. Comparison then proceeds by the rules for comparison of two operands of the class of the comparand.

8.8.4.1.1.5 Comparison of alphanumeric and national operands

Two operands, one class alphanumeric and one class national, may be compared. The alphanumeric operand is treated as though it were converted and moved in accordance with the rules of the MOVE statement to an elementary data item of class national with the same length in terms of character positions as the alphanumeric operand. Comparison then proceeds by the rules for comparison of two operands of class national.

8.8.4.1.1.6 Comparison of alphanumeric operands

An operand of class alphanumeric may be compared to another operand of class alphanumeric or to another operand treated as class alphanumeric for the purposes of comparison. Comparison is made with respect to the collating sequence of characters specified for the current alphanumeric program collating sequence. The length of an operand is the number of alphanumeric character positions in the operand.

Two kinds of comparison are defined: standard comparison and locale-based comparison. Locale-based comparison is used when the alphanumeric program collating sequence in effect is locale based; otherwise, standard comparison is used.

8.8.4.1.1.6.1 Standard comparison

There are two cases to consider: operands of equal length and operands of unequal length.

1) Operands of equal length. Comparison effectively proceeds by comparing alphanumeric characters in corresponding alphanumeric character positions starting from the high-order end and continuing until either

a pair of unequal characters is encountered or the low-order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of corresponding alphanumeric characters are equal. Two zero-length operands are equal.

The first pair of unequal characters encountered are compared to determine their relative position in the alphanumeric collating sequence. The operand that contains the character that is positioned higher in the alphanumeric collating sequence is the greater operand.

2) Operands of unequal length. If the operands are of unequal length, comparison proceeds as though the shorter operand were extended on the right by sufficient alphanumeric spaces to make the operands of equal length. The preceding rule for operands of equal length then apply.

8.8.4.1.1.6.2 Locale-based comparison

For purposes of comparison, trailing spaces are truncated from the operands except that an operand consisting of all spaces is truncated to a single space.

NOTE Locale-based comparison is not necessarily a character-by-character comparison; extending the shorter operand with spaces as for non-locale based comparison could alter the culturally-expected results.

If the locale does not specify a distinct alphanumeric collating sequence, class alphanumeric and alphabetic operands are mapped to their corresponding representation in the national character set for purposes of comparison; the correspondence between alphanumeric characters and national characters is defined by the implementor.

Comparison then proceeds by the algorithm associated with the collating sequence defined by category LC_COLLATE from the current locale. This may be a culturally-sensitive comparison, and is not necessarily performed character-by-character. The determination of whether the relation condition is satisfied is based on the locale specification. Two zero-length operands are equal.

If the locale does not define a collating sequence for all characters of the operands, the EC-LOCALE-INCOMPATIBLE exception condition is set to exist.

8.8.4.1.1.7 Comparison of boolean operands

An operand of class boolean may be compared with another operand of class boolean. Comparison of operands of class boolean is a comparison of their boolean value, regardless of their usage. The length of an operand is the number of boolean positions in the operand. There are two cases to consider: operands of equal length and operands of unequal length.

1) Operands of equal length. Comparison effectively proceeds by comparing boolean values in corresponding boolean positions starting from the leftmost boolean position and continuing until either a pair of unequal boolean values is encountered or the rightmost boolean position of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of corresponding boolean values are equal. Two zero-length operands are equal.

2) Operands of unequal length. If the operands are of unequal length, comparison proceeds as though the shorter operand were extended on the right by sufficient boolean zeros to make the operands of equal length. The preceding rule for operands of equal length then apply.

8.8.4.1.1.8 Comparison of national operands

An operand of class national may be compared with another operand of class national. Comparison is made with respect to the collating sequence of characters specified for the current national program collating sequence. The length of an operand is the number of national character positions in the operand.

Two kinds of comparison are defined: standard comparison and locale-based comparison. Locale-based comparison is used when the national program collating sequence in effect is locale based; otherwise, standard comparison is used.

NOTE An alphanumeric and a national data item may be compared. The rules for comparison of alphanumeric and national operands specify that the alphanumeric operand is converted to national. Comparison then proceeds by the rules for comparison of two national items.

8.8.4.1.1.8.1 Standard comparison

There are two cases to consider: operands of equal length and operands of unequal length.

1) Operands of equal length. Comparison effectively proceeds by comparing national characters in corresponding national character positions starting from the high-order end and continuing until either a pair of unequal characters is encountered or the low-order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of corresponding national characters are equal. Two zero-length operands are equal.

   The first pair of unequal characters encountered are compared to determine their relative position in the national collating sequence. The operand that contains the character that is positioned higher in the national collating sequence is the greater operand.

2) Operands of unequal length. If the operands are of unequal length, comparison proceeds as though the shorter operand were extended on the right by sufficient national spaces to make the operands of equal length. The preceding rule for operands of equal length then apply.

8.8.4.1.1.8.2 Locale-based comparison

For purposes of comparison, trailing spaces are truncated from the operands except that an operand consisting of all spaces is truncated to a single space.

NOTE Locale-based comparison is not necessarily a character-by-character comparison; extending the shorter operand with spaces as for non-locale based comparison could alter the culturally-expected results.

Comparison then proceeds by the algorithm associated with the collating sequence defined by category LC_COLLATE from the current locale. This may be a culturally-sensitive comparison, and is not necessarily performed character-by-character. The determination of whether the relation condition is satisfied is based on the locale specification. Two zero-length operands are equal.

If the locale does not define a collating sequence for all characters of the operands, the EC-LOCALE-INCOMPATIBLE exception condition is set to exist.

8.8.4.1.1.9 Comparison of strongly-typed group items

When two strongly-typed group items are compared, each elementary item of the first operand is compared with the corresponding elementary item of the second operand, in accordance with the rules for comparison of elementary items and in the order in which the elementary items are specified in the strongly-typed group items.

This comparison proceeds until a pair of elementary items is unequal or the final pair of elementary items is compared. The operand that contains the elementary item that is greater than the corresponding elementary item is determined to be the greater operand. Two strongly-typed group operands are determined to be equal if all pairs of corresponding elementary items are equal.

8.8.4.1.1.10 Comparisons involving index-names or index data items

Relation tests may be made only between

1) two index-names. The result is the same as if the corresponding occurrence numbers were compared.

2) an index-name and a numeric data item or numeric literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

3) an index data item and an index-name or another index data item. The actual values are compared without conversion.

8.8.4.1.1.11 Comparisons of operands of class object

An operand of class object may be compared with another operand of class object.

NOTE Comparison of predefined object references with themselves is allowed, although it does not make much sense to do so.

The relation 'identifier-3 = identifier-4' has a true value if the object referenced by identifier-3 is the same object that is referenced by identifier-4; otherwise, the relation has a false value.

8.8.4.1.1.12 Comparison of pointer operands

The operands are equal if they reference the same address.

8.8.4.1.1.13 Comparison of a variable-length group with a compatible group

A comparison of two compatible groups, one or both of which is a variable-length group, proceeds from left to right as described under 8.8.4.1.1.6, Comparison of alphanumeric operands except that:

— when corresponding tables are encountered, they are compared as described in 8.5.1.6.3.5.2, Comparing two tables

— when corresponding any-length elementary items are encountered, the length is determined as described in 8.5.1.7.3, Operations on any-length elementary items.

After comparison of corresponding tables or any-length elementary items, comparison continues with the next data item in each of the compatible groups. Determination of compatible groups and identification of corresponding tables and any-length elementary items is described in 8.5.1.9, Variable-length groups.

8.8.4.1.2 Boolean condition

A boolean condition determines whether a boolean expression is true or false.

8.8.4.1.2.1 General format

[NOT] boolean-expression-1

8.8.4.1.2.2 Syntax rules

1) Boolean-expression-1 shall reference only boolean items of length 1.

8.8.4.1.2.3 General rules

1) Boolean-expression-1 evaluates true if the result of the expression is 1 and evaluates false if the result of the expression is 0.

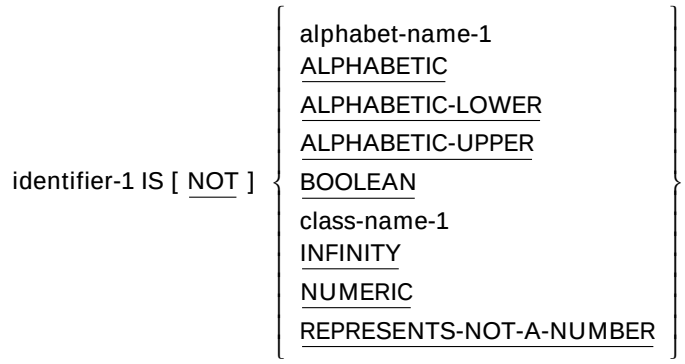
2) The condition NOT boolean-expression-1 evaluates to the reverse truth-value of boolean-expression-1.

8.8.4.1.3 Class condition

The class condition determines whether an operand is numeric, alphabetic, alphabetic-lower, alphabetic-upper, boolean, contains a numeric representation of infinity, or contains only the characters in the set of characters

specified by the ALPHABET or CLASS clause as defined in the SPECIAL-NAMES paragraph of the environment division.

8.8.4.1.3.1 General format



8.8.4.1.3.2 Syntax rules

1) Identifier-1 shall not reference a data item of class index, object, or pointer or a variable-length group.

2) Alphabet-name-1 shall not reference an alphabet associated with a locale.

3) ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, or class-name-1 shall not be specified if the category of the data item referenced by identifier-1 is boolean, numeric, or numeric-edited.

4) BOOLEAN shall not be specified if the category of the data item referenced by identifier-1 is numeric or numeric-edited.

5) If the NUMERIC phrase is specified, identifier-1 shall reference a data item whose usage is display or national or whose category is numeric.

6) If neither the NUMERIC phrase, the INFINITY phrase, nor the REPRESENTS-NOT-A-NUMBER phrase is specified, identifier-1 shall reference a data item whose usage is display or national. If identifier-1 is a function-identifier, it shall reference an alphanumeric or national function.

7) If the INFINITY or REPRESENTS-NOT-A-NUMBER phrase is specified, identifier-1 shall reference a data item of an IEEE floating-point usage.

8.8.4.1.3.3 General rules

1) If the data item referenced by identifier-1 is a zero-length item, the truth value of the class condition without the word NOT is false.

2) If the word NOT is specified, the truth value is reversed.

3) If the data item referenced by identifier-1 is not a zero-length item, the truth value of the class condition without the word NOT is determined as follows:

   a) If alphabet-name-1 is specified, the condition is true if the content of the data item referenced by identifier-1 consists entirely of characters in the coded character set identified by alphabet-name-1 in the SPECIAL-NAMES paragraph.

   b) If ALPHABETIC is specified, the condition is true in the following circumstances:

1. If a locale is in effect for character classification, the condition is true if the content of the data item referenced by identifier-1 consists only of characters identified as alphabetic in locale category LC_CTYPE of the current locale.

2. If a locale is not in effect for character classification, the condition is true if the content of the data item referenced by identifier-1 consists only of a combination of the uppercase letters A, B, C, ..., Z, and space; or a combination of the lowercase letters a, b, c, ..., z and space; or any combination of the uppercase letters and lowercase letters and space.

c) If ALPHABETIC-LOWER is specified, the condition is true in the following circumstances:

1. If a locale is in effect for character classification, the condition is true if the content of the data item referenced by identifier-1 consists only of characters identified as lowercase alphabetic in locale category LC_CTYPE of the current locale.

2. If a locale is not in effect for character classification, the condition is true if the content of the data item referenced by identifier-1 consists only of a combination of the lowercase letters a, b, c, ..., z, and space.

d) If ALPHABETIC-UPPER is specified, the condition is true in the following circumstances:

1. If a locale is in effect for character classification, the condition is true if the content of the data item referenced by identifier-1 consists only of characters identified as uppercase alphabetic in locale category LC_CTYPE of the current locale.

2. If a locale is not in effect for character classification, the condition is true if the content of the data item referenced by identifier-1 consists only of a combination of the uppercase letters A, B, C, ..., Z, and space.

e) If BOOLEAN is specified, the condition is true if the content of the data item referenced by identifier-1 consists entirely of the boolean values '0' and '1'.

f) If class-name-1 is specified, the condition is true if the content of the data item referenced by identifier-1 consists entirely of the characters listed in the definition of class-name-1 in the SPECIAL-NAMES paragraph.

g) If INFINITY is specified, the condition is true if the content of the data item referenced by identifier-1 is a valid representation of positive infinity or negative infinity

NOTE For data items described with IEEE binary floating-point usages, if both a numeric class condition and an infinity class condition are false, the item is a representation of NaN (not a number) as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary encodings. The sign condition can be used to distinguish positive infinity from negative infinity for these usages.

h) If NUMERIC is specified,

1. If the category of the data item referenced by identifier-1 is numeric,

a. If the usage of the data item referenced by identifier-1 is implicitly or explicitly display or national, the condition is true if the presence or absence of an operational sign in the content of the data item referenced by identifier-1 is in agreement with the data description of identifier-1 and if the content, except for the operational sign, consists entirely of the characters 0, 1, 2, 3, ..., 9. Valid operational signs are defined in 13.18.51, SIGN clause.

b. If the usage of the data item referenced by identifier-1 is any IEEE binary floating-point usage, the condition is true only if the content of the data item referenced by identifier-1 represents a finite value according to the specifications for that usage.

NOTE For IEEE binary floating-point usages, any positive infinity, any negative infinity, and any NaN (not-a-number) representations do not represent finite numeric values and are not numeric.

c. If the usage of the data item referenced by identifier-1 is any IEEE decimal floating-point usage, the condition is true only if the content of the data item referenced by identifier-1 represents a finite value encoded in densely-packed decimal according to the specifications for that usage.

NOTE For IEEE decimal floating-point usages, any positive infinity, any negative infinity, and any NaN (not-a-number) representations do not represent finite numeric values and are not numeric. In addition, for items of such usages, this committee draft International Standard does not support the recognition of information encoded in binary-integer form as numeric.

d. Otherwise, the condition is true if the content of the data item referenced by identifier-1 consists entirely of a valid representation for the usage and, if a PICTURE clause is specified, the numeric value is within the range of values implied by the PICTURE clause.

2. If the category of the data item referenced by identifier-1 is not numeric, the condition is true if the content of the data item referenced by identifier-1 consists entirely of the characters 0, 1, 2, 3, ..., 9.

i) If REPRESENTS-NOT-A-NUMBER is specified, the condition is true if the content of the data item referenced by identifier-1 is a valid representation of a signaling or quiet NaN (Not-a-Number) as described in IEEE Std 754-2008, IEEE standard for Floating-Point Arithmetic 3.4, Binary encodings and 3.5, Decimal encodings.

NOTE For data items of IEEE decimal floating-point usages, a value using the binary encoding of a decimal floating-point value as described in IEEE is detectable by a combination of the numeric class condition, the infinity class condition, and the represents-not-a-number class condition, all of which will return a false condition in that case. A prerequisite for the numeric class condition to evaluate to true for any IEEE decimal floating-point item is that the REPRESENTS-NOT-A-NUMBER condition would evaluate to false for that item.

8.8.4.1.4 Condition-name condition (conditional variable)

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with condition-name-1. A conditional variable is defined in 8.4.3, Condition-name.

8.8.4.1.4.1 General Format

condition-name-1

8.8.4.1.4.2 Rules

1) If condition-name-1 is associated with one or more ranges of values, the conditional variable is tested to determine whether its value is within the specified range or ranges, including the end values.

2) The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

3) The result of the test is true if one of the values corresponding to condition-name-1 equals the value of its associated conditional variable.

8.8.4.1.5 Switch-status condition

A switch-status condition determines the on or off status of an implementor-defined external switch. The switch-name and the on or off value associated with the condition shall be named in the SPECIAL-NAMES paragraph of the environment division.

8.8.4.1.5.1 General format

condition-name-1

8.8.4.1.5.2 Rules

The result of the test is true if the switch is set to the specified position corresponding to condition-name-1.

8.8.4.1.6 Sign condition

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to zero.

8.8.4.1.6.1 General format

arithmetic-expression-1 IS [NOT] { POSITIVE | NEGATIVE | ZERO }

8.8.4.1.6.2 Rules

1) When used, NOT and the next keyword specify one sign condition that defines the algebraic test to be executed for truth value. An operand is positive, if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero. If the operand is described with an IEEE floating-point usage, and the content of the operand would evaluate to false in a numeric class condition, the sign is determined according to the description of the corresponding format in IEEE Std 754-2008, IEEE standard for Floating-Point Arithmetic 3.4, Binary encodings and 3.5, Decimal encodings.

NOTE NOT ZERO is a truth test for a nonzero (positive or negative) value.

8.8.4.1.7 Omitted-argument condition

The omitted-argument condition determines whether an argument was provided to a function, method, or program.

8.8.4.1.7.1 General format

data-name-1 IS [NOT] OMITTED

8.8.4.1.7.2 Syntax rules

1) Data-name-1 shall be a formal parameter defined in the source element in which this condition is specified.

8.8.4.1.7.3 General rules

1) The result of the OMITTED test is true:

   a) if the OMITTED phrase, rather than an identifier or literal, is specified as the argument corresponding to data-name-1 in the statement that activated this program, function, or method; or,

   b) the argument corresponding to data-name-1 was a trailing argument that was omitted from the activating statement; or,

   c) if the argument corresponding to data-name-1 is itself a formal parameter for which the omitted-argument condition is true.

2) When used, NOT and the keyword OMITTED specify one condition to be executed for truth value.

8.8.4.2 Complex conditions

A complex condition is formed by combining simple conditions and/or complex conditions with logical connectors (logical operators 'AND' and 'OR') or by negating these conditions with logical negation (the logical operator 'NOT'). The truth value of a complex condition, whether parenthesized or not, is the truth value that results from the interaction of the stated logical operators on its constituent conditions.

The logical operators and their meanings are:

Logical Operator	Meaning
AND	Logical conjunction; the truth value is true if both of the conjoined conditions are true; false if one or both of the conjoined conditions is false.
OR	Logical inclusive OR; the truth value is true if one or both of the included conditions is true; false if both included conditions are false.
NOT	Logical negation or reversal of truth value; the truth value is true if the condition is false; false if the condition is true.

8.8.4.2.1 Negated conditions

A condition is negated by use of the logical operator 'NOT', which reverses the truth value of the condition to which it is applied. Including a negated condition in parentheses does not change its truth value.

NOTE The truth value of a negated condition is true if the truth value of the condition being negated is false; the truth value of a negated condition is false if the truth value of the condition being negated is true.

8.8.4.2.1.1 General format

NOT condition-1

8.8.4.2.2 Combined conditions

A combined condition results from connecting conditions with one of the logical operators 'AND' or 'OR'.

8.8.4.2.2.1 General format

condition-1 { { AND / OR } condition-2 } ...

8.8.4.2.3 Precedence of logical operators and the use of parentheses

The precedence of logical operators determines the conditions to which logical operators apply, unless the precedence is overridden by explicit parentheses. The order of precedence of logical operators is 'NOT', 'AND', 'OR'. Explicit parentheses in a complex condition alter the order of evaluation of the conditions, as described in 8.8.4.3, Order of evaluation of conditions.

NOTE 'condition-1 OR NOT condition-2 AND condition-3' has the meaning 'condition-1 OR ((NOT condition-2) AND condition-3)'. Parentheses can be used to alter the meaning. For example, '(condition-1 OR (NOT condition-2)) AND condition-3' evaluates differently than 'condition-1 OR NOT condition-2 AND condition-3'.

Table 6, Combinations of conditions, logical operators, and parentheses, indicates the ways in which conditions and logical operators may be combined and parenthesized. There shall be a one-to-one correspondence between left and right parentheses such that each left parenthesis is to the left of its corresponding right parenthesis.

Table 6 — Combinations of conditions, logical operators, and parentheses

Given the following element:	In a conditional expression:		In a left-to-right sequence of elements:	
	May element be first?	May element be last?	Element, when not first, may be immediately preceded by only:	Element, when not last, may be immediately followed by only:
simple-condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	No	No	simple-condition,)	simple-condition, NOT, (
NOT	Yes	No	OR, AND, (	simple-condition, (
(	Yes	No	OR, NOT, AND, (	simple-condition, NOT, (
)	No	Yes	simple-condition,)	OR, AND,)
NOTE The element pair 'OR NOT' is permissible while the pair 'NOT OR' is not permissible; the pair 'NOT (' is permissible while the pair 'NOT NOT' is not permissible.				

8.8.4.2.4 Abbreviated combined relation conditions

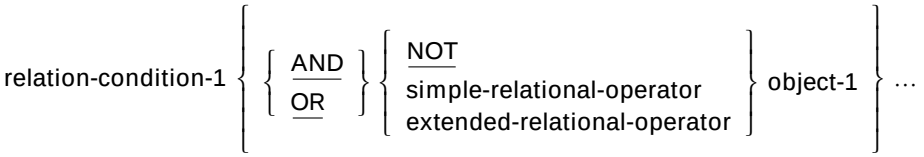
When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that is common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first may be abbreviated by:

1) The omission of the subject of the relation condition, or

2) The omission of the subject and relational operator of the relation condition.

Within a sequence of relation conditions, both forms of omission may be used.

8.8.4.2.4.1 General format



where simple-relational-operator and extended-relational-operator are described in 8.7.5, Relational operators

8.8.4.2.4.2 Syntax Rules

1) Relation-condition-1 shall not be a boolean relation condition.

2) The result of implied insertion shall comply with the rules of table 6, Combinations of conditions, logical operators, and parentheses.

8.8.4.2.4.3 General rules

1) The effect of using abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

NOTE When NOT appears in an abbreviated combined relation condition, it could be as a logical operator preceding the explicit subject of relation-condition-1, or it could be as a part of an extended-relational-operator. However, NOT

cannot immediately precede an extended-relational operator. The interaction of these rules often lead to results that are not intuitive and therefore it should be avoided. Some examples of such usage with the expanded equivalent expression follow:

Abbreviated combined relation condition	Expanded equivalent
a > b AND NOT < c OR d	((a > b) AND (a NOT < c)) OR (a NOT < d)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT a = b OR c	(NOT (a = b)) OR (a = c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))
NOT (a NOT > b AND c AND NOT d)	NOT (((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d)))

8.8.4.3 Order of evaluation of conditions

Parentheses, both explicit and implicit, denote a level of inclusiveness within a complex condition. Two or more conditions connected by only the logical operator 'AND' or only the logical operator 'OR' at the same level of inclusiveness establish a hierarchical level within a complex condition. An entire complex condition can be considered to be a nested structure of hierarchical levels with the entire complex condition itself being the most inclusive hierarchical level. Within this context, the evaluation of the conditions within an entire complex condition begins at the left of the entire complex condition and proceeds according to the following rules recursively applied where necessary:

1) The constituent connected conditions within a hierarchical level are evaluated in order from left to right, and evaluation of that hierarchical level terminates as soon as a truth value for it is determined regardless of whether all the constituent connected conditions within that hierarchical level have been evaluated.

2) Values are established for arithmetic expressions and functions if and when the conditions containing them are evaluated. Similarly, negated conditions are evaluated if and when it is necessary to evaluate the complex condition that they represent. (See 8.8.1, Arithmetic expressions.)

8.9 Reserved words

The following is the list of reserved words:

ACCEPT
ACCESS
ACTIVE-CLASS
ADD
ADDRESS
ADVANCING
AFTER
ALIGNED
ALL
ALLOCATE
ALPHABET
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALSO
ALTERNATE
AND
ANY
ANYCASE
ARE
AREA
AREAS
AS
ASCENDING
ASSIGN
AT

B-AND
B-NOT
B-OR
B-XOR
BASED
BEFORE
BINARY
BINARY-CHAR
BINARY-DOUBLE
BINARY-LONG
BINARY-SHORT
BIT
BLANK
BLOCK
BOOLEAN
BOTTOM
BY

CALL
CANCEL
CF
CH
CHARACTER
CHARACTERS
CLASS
CLASS-ID
CLOSE
CODE
CODE-SET
COL
COLLATING
COLS
COLUMN
COLUMNS
COMMA
COMMON
COMP
COMPUTATIONAL
COMPUTE
CONDITION
CONFIGURATION
CONSTANT
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONVERTING
COPY
CORR
CORRESPONDING
COUNT
CRT
CURRENCY
CURSOR

DATA
DATA-POINTER
DATE
DAY
DAY-OF-WEEK
DE
DECIMAL-POINT
DECLARATIVES
DEFAULT
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

EC
ELSE
END
END-ACCEPT
END-ADD
END-CALL
END-COMPUTE
END-DELETE
END-DISPLAY
END-DIVIDE
END-EVALUATE
END-IF
END-MULTIPLY
END-OF-PAGE
END-PERFORM
END-READ
END-RETURN
END-REWRITE
END-SEARCH
END-START
END-STRING
END-SUBTRACT
END-UNSTRING
END-WRITE
ENVIRONMENT
EO
EOP
EQUAL
ERROR
EVALUATE
EXCEPTION
EXCEPTION-OBJECT
EXIT
EXTEND
EXTERNAL

FACTORY
FALSE
FD
FILE
FILE-CONTROL
FILLER
FINAL
FIRST
FLOAT-BINARY-7
FLOAT-BINARY-16
FLOAT-BINARY-34
FLOAT-DECIMAL-16

FLOAT-DECIMAL-34
FLOAT-EXTENDED
FLOAT-LONG
FLOAT-SHORT
FOOTING
FOR
FORMAT
FREE
FROM
FUNCTION
FUNCTION-ID
FUNCTION-POINTER

GENERATE
GET
GIVING
GLOBAL
GO
GOBACK
GREATER
GROUP
GROUP-USAGE

HEADING
HIGH-VALUE
HIGH-VALUES

I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INFINITY
INHERITS
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INTERFACE
INTERFACE-ID
INTO
INVALID
INVOKE
IS

JUST
JUSTIFIED

KEY

LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCAL-STORAGE
LOCALE
LOCK
LOW-VALUE
LOW-VALUES

MERGE
METHOD
METHOD-ID
MINUS
MODE
MOVE
MULTIPLY

NATIONAL
NATIONAL-EDITED
NATIVE
NEGATIVE
NESTED
NEXT
NO
NOT
NULL
NUMBER
NUMERIC
NUMERIC-EDITED

OBJECT
OBJECT-COMPUTER
OBJECT-REFERENCE
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OPTIONS
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
OVERRIDE

PACKED-DECIMAL
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITIVE
PRESENT
PRINTING
PROCEDURE
PROGRAM
PROGRAM-ID
PROGRAM-POINTER
PROPERTY
PROTOTYPE

QUOTE
QUOTES

RAISE
RAISING
RANDOM
RD
READ
RECORD
RECORDS
REDEFINES
REEL
REFERENCE
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACE
REPLACING
REPORT
REPORTING
REPORTS
REPOSITORY
REPRESENTS-NOT-A-NUMBER
RESERVE
RESET
RESUME
RETRY
RETURN
RETURNING
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED

RUN
SAME
SCREEN
SD
SEARCH
SECTION
SELECT
SELF
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHARING
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SOURCES
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2
START
STATUS
STOP
STRING
SUBTRACT
SUM
SUPER
SUPPRESS
SYMBOLIC
SYNC
SYNCHRONIZED
SYSTEM-DEFAULT

TABLE
TALLYING
TERMINATE
TEST
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
TOP
TRAILING
TRUE
TYPE
TYPEDEF

UNIT
UNIVERSAL
UNLOCK
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USER-DEFAULT
USING

VAL-STATUS
VALID
VALIDATE
VALIDATE-STATUS
VALUE
VALUES
VARYING

WHEN
WITH
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

+
–
*
/
**
>
<
<>
=
>=
<=
&
*>
::
>>

8.10 Context-sensitive words

The following are context-sensitive words and are reserved in the specified language construct or context. If a context-sensitive word is used where the context-sensitive word is permitted in the general format, the word is treated as a keyword; otherwise it is treated as a user-defined word:

Context-sensitive word	Language construct or context
ARITHMETIC	OPTIONS paragraph
ATTRIBUTE	SET statement
AUTO	screen description entry
AUTOMATIC	LOCK MODE clause
AWAY-FROM-ZERO	ROUNDED phrase
BACKGROUND-COLOR	screen description entry
BELL	screen description entry and SET attribute statement
BLINK	screen description entry and SET attribute statement
BYTE-LENGTH	constant entry
CAPACITY	OCCURS clause
CENTER	COLUMN clause
CLASSIFICATION	OBJECT-COMPUTER paragraph
CYCLE	EXIT statement
EOL	ERASE clause in a screen description entry
EOS	ERASE clause in a screen description entry
ENTRY-CONVENTION	OPTIONS paragraph
ERASE	screen description entry
EXPANDS	class-specifier and interface-specifier of the REPOSITORY paragraph
FOREGROUND-COLOR	screen description entry
FOREVER	RETRY phrase
FULL	screen description entry
HIGHLIGHT	screen description entry and SET attribute statement
IGNORING	READ statement
IMPLEMENTS	FACTORY paragraph and OBJECT paragraph
INITIALIZED	ALLOCATE statement and OCCURS clause
INTERMEDIATE	OPTIONS paragraph
INTRINSIC	function-specifier of the REPOSITORY paragraph
LC_ALL	SET statement
LC_COLLATE	SET statement
LC_CTYPE	SET statement
LC_MESSAGES	SET statement
LC_MONETARY	SET statement

Context-sensitive word	Language construct or context
LC_NUMERIC	SET statement
LC_TIME	SET statement
LOWLIGHT	screen description entry and SET attribute statement
MANUAL	LOCK MODE clause
MULTIPLE	LOCK ON phrase
NEAREST-AWAY-FROM-ZERO	INTERMEDIATE ROUNDING clause and ROUNDED phrase
NEAREST-EVEN	INTERMEDIATE ROUNDING clause and ROUNDED phrase
NEAREST-TOWARD-ZERO	INTERMEDIATE ROUNDING clause and ROUNDED phrase
NEGATIVE-INFINITY	SET statement
NONE	DEFAULT clause
NORMAL	STOP statement
NOT-A-NUMBER	SET statement
NUMBERS	COLUMN clause and LINE clause
ONLY	Object-view, SHARING clause, SHARING phrase, and USAGE clause
PARAGRAPH	EXIT statement
POSITIVE-INFINITY	SET statement
PREFIXED	ANY LENGTH STRUCTURE clause
PREVIOUS	READ statement
PROHIBITED	INTERMEDIATE ROUNDING clause and ROUNDED phrase
RECURSIVE	PROGRAM-ID paragraph
RELATION	VALIDATE-STATUS clause
REQUIRED	screen description entry
REVERSE-VIDEO	screen description entry and SET attribute statement
ROUNDING	OPTIONS paragraph
SECONDS	RETRY phrase
SECURE	screen description entry
SHORT	ANY LENGTH STRUCTURE clause
SIGNED	ANY LENGTH STRUCTURE clause and USAGE clause
STANDARD-BINARY	ARITHMETIC clause
STANDARD-DECIMAL	ARITHMETIC clause
STATEMENT	RESUME statement
STEP	OCCURS clause
STRONG	TYPEDEF clause
SYMBOL	CURRENCY clause
TOWARD-GREATER	ROUNDED phrase

Context-sensitive word	Language construct or context
TOWARD-LESSER	ROUNDED phrase
TRUNCATION	INTERMEDIATE ROUNDING clause and ROUNDED phrase
UCS-4	ALPHABET clause
UNDERLINE	screen description entry and SET attribute statement
UNSIGNED	USAGE clause
UTF-8	ALPHABET clause
UTF-16	ALPHABET clause
YYYYDDD	ACCEPT statement
YYYYMMDD	ACCEPT statement

All exception-names are context-sensitive because they may appear only following RAISE, RAISING (in GOBACK, EXIT FUNCTION, EXIT METHOD, and EXIT PROGRAM), USE EXCEPTION, and in the TURN compiler directive. The list of exception-names is given in 14.6.12.1, Exception conditions.

8.11 Intrinsic function names

The following is the list of intrinsic function names:

ABS
ACOS
ANNUITY
ASIN
ATAN
BOOLEAN-OF-INTEGER
BYTE-LENGTH
CHAR
CHAR-NATIONAL
COMBINED-DATETIME
COS
CURRENT-DATE
DATE-OF-INTEGER
DATE-TO-YYYYMMDD
DAY-OF-INTEGER
DAY-TO-YYYYDDD
DISPLAY-OF
E
EXCEPTION-FILE
EXCEPTION-FILE-N
EXCEPTION-LOCATION
EXCEPTION-LOCATION-N
EXCEPTION-STATEMENT
EXCEPTION-STATUS
EXP
EXP10
FACTORIAL
FORMATTED-CURRENT-DATE
FORMATTED-DATE
FORMATTED-DATETIME
FORMATTED-TIME
FRACTION-PART
HIGHEST-ALGEBRAIC
INTEGER
INTEGER-OF-BOOLEAN
INTEGER-OF-DATE
INTEGER-OF-DAY
INTEGER-OF-FORMATTED-DATE
INTEGER-PART
LENGTH
LOCALE-COMPARE
LOCALE-DATE
LOCALE-TIME
LOCALE-TIME-FROM-SECONDS
LOG
LOG10
LOWER-CASE
LOWEST-ALGEBRAIC
MAX
MEAN
MEDIAN
MIDRANGE
MIN
MOD
NATIONAL-OF
NUMVAL
NUMVAL-C
NUMVAL-F
ORD
ORD-MAX
ORD-MIN
PI
PRESENT-VALUE
RANDOM
RANGE
REM
REVERSE
SECONDS-FROM-FORMATTED-TIME
SECONDS-PAST-MIDNIGHT
SIGN
SIN
SQRT
STANDARD-COMPARE
STANDARD-DEVIATION
SUM
TAN
TEST-DATE-YYYYMMDD
TEST-DAY-YYYYDDD
TEST-FORMATTED-DATETIME
TEST-NUMVAL
TEST-NUMVAL-C
TEST-NUMVAL-F
TRIM
UPPER-CASE
VARIANCE
WHEN-COMPILED
YEAR-TO-YYYY

8.12 Compiler-directive words

The following words are reserved in compiler directives:

ALL
AND
AS

B-AND
B-NOT
B-OR
B-XOR

CALL-CONVENTION
CHECKING
COBOL
CORRESPONDING

DE-EDITING
DEFINE
DEFINED
DIVIDE

ELSE
END-EVALUATE
END-IF
EQUAL
EVALUATE

FIXED
FLAG-02
FLAG-85
FLAG-NATIVE-ARITHMETIC
FORMAT
FREE
FUNCTION-ARGUMENT

GREATER

IF
IMP
IS

LEAP-SECOND
LESS
LISTING
LOCATION

MOVE

NOT
NUMVAL

OFF
ON
OR
OTHER
OVERRIDE

PAGE
PARAMETER
PROPAGATE

SET
SIZE
SOURCE
STANDARD-1
STANDARD-2

THAN
THROUGH
THRU
TO
TRUE
TURN

WHEN
WITH

ZERO-LENGTH

+
-
*
/
<=
>=
<
>
<>
=
(
)

In addition to the above list, all of the exception-names specified in 14.6.12.1, Exception conditions, are reserved in the context of compiler directives.

8.13 External repository

The external repository contains all information required for activating programs, functions, or methods and for checking conformance. This information includes:

- the externalized name of the source unit
- the type of the source unit - program, function, class, or interface
- the description of the parameters of the source unit, if any, and the manner of receiving parameters (by reference or by value) and whether they are optional or not
- the description of the returning item of the source unit, if any
- the exceptions that may be raised by the runtime element, if any
- the entry convention of the source unit, if any
- the object properties of the source unit, if any
- the methods contained in the source unit, if any, and details about the method's externalized name, parameters, returning item, and entry convention
- type declarations required for the description of parameters and returning items
- whether the DECIMAL-POINT IS COMMA clause is specified in the source unit
- any currency symbols and their corresponding currency strings defined in the source unit
- any external locale identification for locales associated with formal parameters or returning items of the source unit
- any other information that the implementor requires

This information about a source unit, excluding the externalized name of the source unit, is called its signature. Whether the information is taken from a prototype or a definition, the information stored in the external repository about the signature of a program or a function is the same.

The implementor shall provide a mechanism that allows the user to specify whether to update the external repository when a compilation unit is compiled.

The implementor shall provide a mechanism that allows the user to specify whether to flag differences in prototypes and definitions in the compilation group from the information in the external repository.

The details on the association of the name of a source unit with information in the external repository are specified in 12.3.7, REPOSITORY paragraph.

9 I-O, objects, and user-defined functions

9.1 Files

9.1.1 Physical and logical files

The physical aspects of a file describe the data as it appears on the input or output media and include such features as:

1) The grouping of logical records within the physical limitations of the file medium.

2) The means by which the file shall be identified.

The conceptual, or logical, characteristics of a file are the explicit definition of each logical entity within the file itself. COBOL input or output statements refer to one logical record at a time.

It is important to distinguish between a physical record and a logical record. A COBOL logical record is a group of related information, uniquely identifiable, and treated as a unit.

A physical record is a physical unit of information transferred to or recorded on an output device or transferred from an input device. The size of a physical record is hardware dependent and bears no direct relationship to the size of the file of information contained on a device.

A logical record may be contained within a single physical unit; or several logical records may be contained within a single physical unit; or a logical record may require more than one physical unit to contain it. There are several source language methods available for describing the relationship of logical records and physical units. When a permissible relationship has been established, control of the accessibility of logical records as related to the physical unit is provided by the interaction of the runtime module with the processor. In this document, references to records mean to logical records, unless the term 'physical record' is specifically used. Similarly, references to files mean to the logical characteristics of a file, unless 'physical file' is used. For each file connector there is one associated logical file that is referenced by the file-name that refers to that file connector, even though there may be several logical files associated with one physical file.

When a logical record is transferred to or from a physical unit, any translation required by the presence of a CODE-SET or FORMAT clause is accomplished. Padding characters are added or deleted as necessary.

9.1.2 Record area

The record area is a storage area associated with a file in which logical records from that file are made accessible to a runtime element. The record area is made accessible to the runtime element at the completion of a successfully executed OPEN statement. For files open in the input mode, the logical record is available in the record area after execution of a successful read. For files open in the extend or output mode, the logical record is available until execution of a successful write or rewrite. For files open in the I-O mode, the logical record is available after execution of a successful read until the execution of either a read or successful rewrite.

NOTE All record description entries subordinate to a file description entry (FD or SD) implicitly redefine the same storage area, as specified in 13.18.32, Level-number, general rule 3.

9.1.3 File connector

A file connector is referenced by a file-name and it is a storage area that is not visible to the user that contains information used by the run unit to determine the status of input-output operations and of the connection to the physical file.

A file connector has several attributes that are specified by phrases and clauses in the file description entry and the file control entry for the associated file-name and by the execution of input-output statements. These attributes are: organization (sequential, indexed, or relative); access mode (sequential, dynamic, or random); lock mode (automatic, manual, or none); locking mode (single record locking, multiple record locking, or none); in an open

mode (input, output, i-o, extend); sharing mode (sharing with no other, sharing with read only, sharing with all other, implementor-defined, or no sharing); and whether or not it is a report file connector. It also contains information about the file position indicator, the key of reference, the I-O status value, the current volume pointer, and file and record locks.

A file connector is either internal or external as described in 8.6.2, External and internal items. For internal file connectors, one file connector is associated with each file description entry. For external file connectors, there is only one file connector that is associated with the run unit no matter how many file description entries describe the same file-name.

9.1.4 Open mode

A file connector is open when its open mode is either input, output, i-o, or extend. When a file connector is open, it is the linkage between the logical file and the physical file.

A file connector is placed in an open mode by the execution of a successful OPEN statement that references the associated file-name. The OPEN statement also associates the file connector with a physical file. When a CLOSE statement references the associated file-name, the file connector is no longer associated with the physical file and the file connector is no longer in an open mode. In the following cases, the COBOL runtime system executes an implicit CLOSE statement without any optional phrases for a file connector that is in the open mode:

— When the run unit terminates.

— For initial file connectors described in a program when a GOBACK or an EXIT PROGRAM statement is executed in a called program in which they are described.

— For file connectors in the program to which a CANCEL statement is executed or in any program contained in that program.

— For file connectors in an object when the object is deleted.

9.1.5 Sharing file connectors

Two runtime elements in a run unit may reference common file connectors in the following circumstances:

1) An external file connector may be referenced from any runtime element that describes that file connector.

2) If a program is contained within another program, both programs may refer to a common file connector by referring to an associated global file-name either in the containing program or in any program that directly or indirectly contains the containing program.

9.1.6 Fixed file attributes

A physical file has several attributes that apply to the file at the time it is created and cannot be changed throughout the lifetime of the file. The primary attribute is the organization of the file, that describes its logical structure. There are three organizations: sequential, relative, and indexed. Other fixed attributes of the physical file provided through COBOL are prime record key, alternate record keys, code set, the minimum and maximum logical record size in bytes, the record type (fixed or variable), the collating sequence of the keys for indexed files, the minimum and maximum physical record size in bytes, and the record delimiter. The implementor shall specify whether the ability to share a physical file is a fixed file attribute.

9.1.7 Organization

There are three file organizations: sequential, relative, and indexed.

9.1.7.1 Sequential

Sequential files are organized so that each record, except the last, has a unique successor record; each record, except the first, has a unique predecessor record. The successor relationships are established by the order of execution of WRITE statements when the physical file is created. Once established, successor relationships do not change except in the case where records are added to the end of a physical file.

A sequential physical file that is on a mass storage device has the same logical structure as a physical file on any sequential medium; however, logical records that are mapped to physical records on a sequential physical file on a mass storage device may be updated in place. When this technique is used, the replacing physical record shall have the same size as the original physical record.

9.1.7.2 Relative

Relative files are organized so that each record may be stored or retrieved by providing the value of the record's relative record number. A relative file shall be associated with a relative physical file that is on a mass storage device.

Conceptually, a file with relative organization is a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number. Each logical record in a relative file is identified by the relative record number of its storage area.

> NOTE For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in any of the first through the ninth record areas.

In order to achieve more efficient access to records in a relative file, the number of bytes reserved in a physical record on the mass storage device to store a particular logical record may be different from the number of bytes in the description of that record in the data division.

9.1.7.3 Indexed

Indexed files are organized so that each record may be stored, retrieved, or deleted by providing the value of a specified key in that record. An indexed file shall be associated with an indexed physical file that is on a mass storage device. For each key data item defined for the records of a file, an index is maintained. Each such index represents the set of values from the corresponding key data item in each record. Each index is a mechanism that provides access to any record in the file.

Each indexed file has a primary index that represents the prime record key of each record in the file. Each record is inserted in the file, changed, or deleted from the file based solely upon the value of its prime record key. The prime record key of each record in the file shall be unique, and it shall not be changed when updating a record. The prime record key is declared in the RECORD KEY clause of the file control entry for the file.

Alternate record keys provide alternate means of retrieval for the records of a file. Such keys are named in the ALTERNATE RECORD KEY clause of the file control entry. When the DUPLICATES phrase is specified in the ALTERNATE RECORD KEY clause, the value of a particular alternate record key need not be unique within the file.

Both the prime record and any alternate record keys are made up from one or more portions of the record area associated with the file. For each key, the number of such components, their length, and their relative positions within the record area are fixed file attributes and shall not be changed once the physical file has been created.

9.1.8 Access modes

The ACCESS MODE clause of the file description entry specifies the manner in which the runtime element operates upon records within a file. The access mode may be sequential, random, or dynamic.

For files that are organized as relative or indexed, any of the three access modes may be used to access the file regardless of the access mode used to create the physical file. A file with sequential organization may be accessed only in sequential mode.

9.1.8.1 Sequential access mode

For sequential organization the order of sequential access when NEXT is specified or implied on a READ statement is the order in which the records were originally written to the physical file. When PREVIOUS is specified on a READ statement the order is the reverse of the order in which the records were originally written. The START statement can be used to position the file at the beginning or the end for subsequent retrievals.

For relative organization the order of sequential access when NEXT is specified or implied on a READ statement is ascending based on the value of the relative record number. When PREVIOUS is specified on a READ statement the order is descending based on the value of the relative record number. The START statement can be used to establish a starting point for a series of subsequent sequential retrievals, either in a forward or reverse direction.

For indexed organization the order of sequential access when NEXT is specified or implied on a READ statement is ascending based on the value of the key of reference according to the collating sequence of the physical file. Any of the keys associated with the file may be established as the key of reference during the processing of the file. The order of retrieval from a set of records that have duplicate key of reference values is the original order of arrival of those records into that set. When PREVIOUS is specified on a READ statement the order is descending based on the value of the key of reference according to the collating sequence of the physical file. The order of retrieval from a set of records that have duplicate key of reference values is the reverse of the original order of arrival of those records into that set. The START statement can be used to establish a starting point for a series of subsequent sequential retrievals, either in a forward or reverse direction.

9.1.8.2 Random access mode

When a file is accessed in random mode, input-output statements are used to access the records in a programmer-specified order. The random access mode may be used only with relative or indexed file organizations. For a file with relative organization, the programmer specifies the desired record by placing its relative record number in a relative key data item. With the indexed organization, the programmer specifies the desired record by placing the value of one of its record keys in a record key or an alternate record key data item.

9.1.8.3 Dynamic access mode

With dynamic access mode, the programmer may change at any time between sequential access and random access, using appropriate forms of input-output statements. The dynamic access mode may be used only on files with relative or indexed organizations.

9.1.9 Reel and unit

The terms 'reel' and 'unit' are synonymous. They are applicable only to files with sequential organization that are associated with a physical file that may be contained on multiple physical devices. Treatment of such files is logically equivalent to the treatment of a sequential file that is associated with a physical file that is wholly contained on one physical device.

NOTE An example of a physical file stored on multiple physical devices is a physical file that is contained on multiple magnetic tapes. Another is one that is stored on multiple removable disk packs.

9.1.10 Current volume pointer

The current volume pointer is a conceptual entity used in this document to facilitate exact specification of the current physical volume of a sequential file. The status of the current volume pointer is affected by the CLOSE, OPEN, READ, and WRITE statements.

9.1.11 File position indicator

The file position indicator is a conceptual entity that exists for each file connector that is open in the i-o mode or input mode, and is used to facilitate exact specification of the record to be accessed during certain sequences of input-output operations. The setting of the file position indicator is affected only by the CLOSE, OPEN, READ, and START statements.

The file position indicator contains the value of the current key within the key of reference for an indexed file, the record number of the current record for a sequential file, the relative record number of the current record for a relative file, or indicates one of the following conditions for the file connector:

1) No valid record position has been established.

2) An optional file is not present.

3) No next or previous logical record exists.

9.1.12 I-O status

The I-O status is a two-character conceptual entity whose value is set to indicate the status of an input-output operation during the execution of a CLOSE, DELETE, OPEN, READ, REWRITE, START, UNLOCK or WRITE statement and prior to the execution of any imperative statement associated with that input-output statement or prior to the execution of any applicable USE EXCEPTION procedure. The value of the I-O status is made available through the use of the FILE STATUS clause in the file control entry for the file or through the use of the EXCEPTION-FILE or EXCEPTION-FILE-N function.

The I-O status also determines whether an applicable USE EXCEPTION procedure is executed. If any condition occurs other than those specified in 9.1.12.1, Successful completion, execution of such a procedure is dependent on rules stated elsewhere. No such procedure is executed if one of the conditions specified in 9.1.12.1, Successful completion, occurs.

Certain classes of I-O status values indicate fatal exception conditions. These are: any that begin with the digit 3 or 4, and any that begin with the digit 9 that the implementor defines as fatal. If the value of the I-O status for an input-output operation indicates a fatal exception condition, the implementor determines what action is taken after the execution of any applicable USE EXCEPTION procedure, or if none applies, after completion of the normal input-output control system error processing. The implementor may either continue or terminate the execution of the run unit. If the implementor chooses to continue execution of the run unit, control is transferred to the end of the statement that produced the fatal exception condition unless the rules for that statement define other behavior. Any NOT AT END or NOT INVALID KEY phrase specified for that statement is ignored.

Any I-O status associated with an unsuccessful completion is associated with an exception condition. Whether or not the exception condition is raised depends on whether or not checking for that exception condition is enabled. The exception condition depends on the first character of the I-O status value that results after the execution of an input-output statement. The exception-names and the first digit of their corresponding I-O status values are:

EC-I-O-AT-END	'1'
EC-I-O-INVALID-KEY	'2'
EC-I-O-PERMANENT-ERROR	'3'
EC-I-O-LOGIC-ERROR	'4'
EC-I-O-RECORD-OPERATION	'5'
EC-I-O-FILE-SHARING	'6'
EC-I-O-IMP	'9'

If the first character of the resulting I-O status value is one of the above values, the associated exception condition is set to exist. If the exception condition EC-I-O-AT-END or EC-I-O-INVALID-KEY exists and the input-output statement that caused the exception condition to exist is specified with an AT END or INVALID KEY phrase respectively, no USE procedure shall be executed. Otherwise, the exception condition that exists determines whether an applicable USE procedure shall be executed according to the rules in 14.9.45, USE statement.

I-O status expresses one of the following conditions upon completion of the input-output operation:

1) Successful completion. The input-output statement was successfully executed.

2) Implementor-defined successful completion. A condition specified by the implementor occurred and the input-output statement was successfully executed.

3) At end. A sequential READ statement was unsuccessfully executed as a result of an at end condition.

4) Invalid key. The input-output statement was unsuccessfully executed as a result of an invalid key condition.

5) Permanent error. The input-output statement was unsuccessfully executed as the result of an error that precluded further processing of the file. Any specified exception procedures are executed. The permanent error condition remains in effect for all subsequent input-output operations on the file unless an implementor-defined technique is invoked to correct the permanent error condition.

6) Logic error. The input-output statement was unsuccessfully executed as a result of an improper sequence of input-output operations that were performed on the file or as a result of violating a limit defined by the user.

7) Record operation conflict. The input-output statement was unsuccessfully executed as a result of the record being locked by another file connector.

8) File sharing conflict. The input-output statement was unsuccessfully executed as a result of the file being locked by another file connector.

9) Implementor-defined unsuccessful completion. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the implementor.

9.1.12.1 through 9.1.12.9 specify the values placed in the I-O status for the previously named conditions resulting from the execution of an input-output operation. If more than one value applies, the implementor determines which of the applicable values to place in the I-O status.

9.1.12.1 Successful completion

1) I-O status = 00. The input-output statement is successfully executed and no further information is available concerning the input-output operation.

2) I-O status = 02. The input-output statement is successfully executed but a duplicate key is detected.

   a) For a READ statement with the NEXT phrase specified or implied, the key value for the current key of reference is equal to the value of the same key in the next record in the physical file.

   b) For a READ statement with the PREVIOUS phrase specified, the key value for the current key of reference is equal to the value of the same key in the prior record in the physical file.

   c) For a REWRITE or WRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.

3) I-O status = 04. A READ statement is successfully executed but the length of the record being processed does not conform to the fixed file attributes for that file.

4) I-O status = 05. An OPEN statement is successfully executed but the file is described as optional and the physical file is not present at the time the OPEN statement is executed. If the open mode is I-O or extend, the physical file has been created.

5) I-O status = 07. The input-output statement is successfully executed but a CLOSE statement with the NO REWIND, REEL/UNIT, or FOR REMOVAL phrase or for an OPEN statement with the NO REWIND phrase references a physical file on a non-reel/unit medium.

9.1.12.2 Implementor-defined successful completion

1) I-O status = 0x. An implementor-defined condition exists. The value of x is specified by the implementor and may be any of the uppercase letters 'A' through 'M' or lowercase letters 'a' through 'm', where the range of letters is defined by the sequence of letters in ISO/IEC 646. This condition shall not duplicate any condition specified for I-O status values 9x or 00 through 61.

9.1.12.3 At end condition with unsuccessful completion

1) I-O status = 10. A sequential READ statement is attempted and no next or prior logical record exists in the physical file because:

   a) NEXT was specified or implied and the end of the physical file has been reached, or

   b) PREVIOUS was specified and the beginning of the physical file has been reached, or

   c) a sequential READ statement is attempted for the first time on a file described as optional and the physical file is not present.

2) I-O status = 14. A sequential READ statement is attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

9.1.12.4 Invalid key condition with unsuccessful completion

1) I-O status = 21. A sequence error exists for a sequentially accessed indexed file. The prime record key value has been changed by the runtime element between the successful execution of a READ statement through a file connector and the execution of the next REWRITE statement for that file through the same file connector, or the ascending sequence requirements for successive record key values are violated. (See 14.9.47, WRITE statement.)

2) I-O status = 22. An attempt is made either:

   a) to write a record that would create a duplicate key in a physical relative file.

   b) to write a record that would create a duplicate prime record key in a physical indexed file, or

   c) to write or rewrite a record that would create a duplicate alternate record key when the DUPLICATES phrase is not specified for that alternate record key in the physical file.

3) I-O status = 23. This condition exists because:

   a) an attempt is made to randomly access a record that does not exist in the physical file; or

   b) a START or random READ statement is attempted on a file described as optional and the physical file is not present; or

   c) a START statement is attempted with an invalid key length specification; or

   d) a START statement is attempted on a sequential file that has no records or that does not support the ability to position at the specified record.

4) I-O status = 24. An attempt is made to write beyond the externally-defined boundaries of a physical relative or indexed file. The implementor specifies the manner in which these boundaries are defined. Or, a sequential WRITE statement is attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

9.1.12.5 Permanent error condition with unsuccessful completion

1) I-O status = 30. A permanent error exists and no further information is available concerning the input-output operation.

2) I-O Status = 31. A permanent error exists during execution of an OPEN statement because the content of the data item referenced by the data-name specified in the USING phrase of the file control entry is not consistent with the specification for the device-name or literal in the ASSIGN clause of that file control entry.

3) I-O status = 34. A permanent error exists because of a boundary violation; an attempt is made to write beyond the externally-defined boundaries of a physical sequential file. The implementor specifies the manner in which these boundaries are defined.

4) I-O status = 35. A permanent error exists because an OPEN statement with the INPUT, I-O, or EXTEND phrase is attempted on a file that is not described as optional and the physical file is not present.

5) I-O status = 37. A permanent error exists because an OPEN statement is attempted on a file and that file will not support the open mode specified in the OPEN statement. The possible violations are:

   a) the EXTEND or OUTPUT phrase is specified but the file will not support write operations.

   b) the I-O phrase is specified but the file will not support the input and output operations that are permitted for the organization of that file when opened in the I-O mode.

   c) the INPUT phrase is specified but the file will not support read operations.

6) I-O status = 38. A permanent error exists because an OPEN statement is attempted on a file connector previously closed with lock.

7) I-O status = 39. The OPEN statement is unsuccessful because a conflict has been detected between the fixed file attributes and the attributes specified for that file in the source unit.

9.1.12.6 Logic error condition with unsuccessful completion

1) I-O status = 41. An OPEN statement is attempted for a file connector in an open mode.

2) I-O status = 42. A CLOSE or UNLOCK statement is attempted for a file connector that is not in an open mode.

3) I-O status = 43. For a mass storage file in the sequential access mode, the last input-output statement executed for the associated file through a file connector prior to the execution of a DELETE or REWRITE statement through the same file connector was not a successfully executed READ statement.

4) I-O status = 44. A boundary violation exists because:

   a) an attempt is made to write or rewrite a record that is larger than the largest or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name, or

   b) an attempt is made to rewrite a record to a sequential file and the record is not the same size as the record being replaced.

   c) an attempt is made to write or rewrite a record that is larger than the largest or smaller than the smallest record allowed by the fixed-or-variable-length format of the RECORD clause when the implementor has specified that variable-length records are produced.

5) I-O status = 45. Record identification failure. The input-output statement was unsuccessful because no record description entry was selected for processing with the FORMAT clause or the CODE-SET clause.

6) I-O status = 46. A sequential READ statement is attempted referencing a file connector open in the input or I-O mode and no valid next record has been established because:

   a) The preceding START statement referencing that file connector was unsuccessful, or

   b) The preceding READ statement referencing that file connector was unsuccessful.

7) I-O status = 47. The execution of a READ or START statement is attempted referencing a file connector that is not open in the input or I-O mode.

8) I-O status = 48. The execution of a WRITE statement is attempted referencing a file connector that is not open in the correct open mode as follows:

   a) If the access mode is sequential, the file connector is not open in the extend or output mode.

   b) If the access mode is dynamic or random, the file connector is not open in the I-O or output mode.

9) I-O status = 49. The execution of a DELETE or REWRITE statement is attempted referencing a file connector that is not open in the I-O mode.

9.1.12.7 Record operation conflict condition with unsuccessful completion

1) I-O status = 51. The input-output statement is unsuccessful due to an attempt to access a record that is currently locked by another file connector.

2) I-O status = 52. The input-output statement is unsuccessful due to a deadlock. The implementor shall specify under what conditions a deadlock is detected.

3) I-O status = 53. The input-output statement is unsuccessful because the statement requested a record lock, but this run unit holds the maximum number of locks allowed by this implementation.

4) I-O status = 54. The input-output statement is unsuccessful because the statement requested a record lock, but this file connector holds the maximum number of locks allowed by this implementation.

9.1.12.8 File sharing conflict condition with unsuccessful completion

1) I-O status = 61. A file sharing conflict condition exists because an OPEN statement is attempted on a physical file and that physical file is already open by another file connector in a manner that conflicts with this request. The possible violations are:

   a) An attempt is made to open a physical file that is currently open by another file connector in the sharing with no other mode.

   b) An attempt is made to open a physical file in the sharing with no other mode and the physical file is currently open by another file connector.

   c) An attempt is made to open a physical file for I-O or extend and the physical file is currently open by another file connector in the sharing with read only mode.

   d) An attempt is made to open a physical file in the sharing with read only mode and the physical file is currently open by another file connector in the I-O or extend mode.

   e) An attempt is made to open a physical file in the output mode and the physical file is currently open by another file connector.

9.1.12.9 Implementor-defined condition with unsuccessful completion

1) I-O status = 9x. An implementor-defined condition exists. This condition shall not duplicate any condition specified by the I-O status values 00 through 61. The value of x is defined by the implementor.

9.1.13 Invalid key condition

The invalid key condition may occur as a result of the execution of a DELETE, READ, REWRITE, START, or WRITE statement. When the invalid key condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected.

If the invalid key condition exists after the execution of the input-output operation specified in an input-output statement, the following actions occur in the order shown:

1) The I-O status of the file connector associated with the statement is set to a value indicating the invalid key condition.

2) If the INVALID KEY phrase is specified in the input-output statement, any USE EXCEPTION file procedure associated with the file connector or any USE EXCEPTION EC-I-O-INVALID-KEY procedure, if that exception condition was raised, is not executed and control is transferred to the imperative-statement specified in the INVALID KEY phrase. Execution then continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the INVALID KEY phrase, control is transferred to the end of the input-output statement and the NOT INVALID KEY phrase, if specified, is ignored.

3) If the INVALID KEY phrase is not specified in the input-output statement and a USE AFTER EXCEPTION procedure is associated with the file connector associated with the input-output statement, that USE AFTER EXCEPTION procedure is executed and control is transferred according to the rules of the USE statement. The NOT INVALID KEY phrase is ignored, if it is specified.

4) If the INVALID KEY phrase is not specified in the input-output statement and there is no USE AFTER EXCEPTION procedure associated with the file connector associated with the input-output statement, control is transferred to the end of the input-output statement. The NOT INVALID KEY phrase is ignored, if it is specified.

If the invalid key condition does not exist after the execution of the input-output operation specified by an input-output statement, the INVALID KEY phrase is ignored, if specified. The I-O status of the file connector associated with the statement is updated and the following actions occur:

1) If the I-O status indicates an unsuccessful completion that is not an invalid key condition, control is transferred according to the rules of any USE EXCEPTION file procedure associated with the file connector or USE EXCEPTION exception-name procedure associated with an EC-I-O exception condition that was raised.

2) If the I-O status indicates a successful completion, control is transferred to the end of the input-output statement or to the imperative-statement specified in the NOT INVALID KEY phrase if it is specified. In the latter case, execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the NOT INVALID KEY phrase, control is transferred to the end of the input-output statement.

9.1.14 Sharing mode

The sharing mode indicates whether a file is to participate in file sharing and record locking, and specifies the degree of file sharing (or non-sharing) to be permitted for the file. The sharing mode specifies the types of operations that may be performed on the shared physical file through other file connectors throughout the duration of this OPEN.

The SHARING phrase on an OPEN statement overrides the SHARING clause in the file control entry for establishing the sharing mode. If there is no SHARING phrase on the OPEN statement, the sharing mode is completely determined by the SHARING clause in the file control entry. If no specification is made in either location, the implementor defines the sharing mode in which the file is opened; the implementor-defined sharing mode may be one of the modes specified in this final committee draft International Standard or may be a mode completely specified by the implementor. The rules are the same for a given standard sharing mode regardless of whether the sharing mode is specified on the OPEN statement, specified in the file control paragraph, or specified as the default by the implementor.

Other facilities may specify some degree of file sharing, however, their interaction with COBOL file sharing is defined by the implementor.

NOTE These facilities can include a job control language or another programming language. Implementors are encouraged to honor file and record locks in a multi-language environment. The implementor should document the file sharing and record locking facility in a way that programs written in other languages might make use of it, if it is possible for these programs to do so.

A shared physical file shall reside on a device that allows concurrent access to the file. The implementor shall specify which devices allow concurrent access to a physical file.

Before access to a shared physical file is allowed through an OPEN statement, the sharing mode and the open mode shall be allowed by all other file connectors that are currently associated with the physical file. Additionally, the sharing mode for the current OPEN statement shall permit all of the sharing modes and open modes that exist for all other file connectors that are currently associated with the physical file. (See 9.1.12, I-O status; 14.9.26, OPEN statement; and table 20, Opening available shared files that are currently open by another file connector.)

The sharing mode controls access to a physical file as follows:

1) The sharing with no other mode specifies exclusive access to a physical file. Associating this file connector with the physical file will be unsuccessful if the physical file is currently open through other file connectors. If the OPEN statement is successful, subsequent requests to open the physical file through other file connectors before this file connector is closed will be unsuccessful. Record locks are ignored.

2) The sharing with read only mode restricts concurrent access to a physical file through file connectors other than this one, to input mode. Associating this file connector with the physical file will be unsuccessful if the physical file is associated with another file connector whose open mode is other than input. If the OPEN statement is successful, subsequent requests to open the physical file through other file connectors in a mode other than input before this file connector is closed will be unsuccessful. Record locks are in effect.

3) The sharing with all other mode allows concurrent access to a physical file through other file connectors specifying input, I-O, or extend mode, subject to any further restrictions that apply. Record locks are in effect.

Multiple paths of access may exist in the same runtime element, contained elements, separate runtime elements within the same run unit, or runtime elements in different run units.

The setting of a file lock is part of the atomic operation of an I-O statement.

The file lock is removed by an explicit or implicit CLOSE statement executed for that file connector.

9.1.15 Record locking

Record locking provides the capability of controlling concurrent access to logical records in a shared file. Two modes of locking are available, AUTOMATIC and MANUAL. Single-record locking or multiple-record locking is available for both AUTOMATIC and MANUAL locking.

For automatic single-record locking, the runtime system controls the setting and releasing of locks. For automatic multiple-record locking, the runtime system controls the setting of locks, and the application controls the releasing of locks by the execution of an explicit UNLOCK statement.

For manual single-record and multiple-record locking, the application controls the setting and releasing of locks by the use of locking phrases on input-output statements and the use of the UNLOCK statement.

While locked by a given file connector, a record is not accessible to another file connector in the same or a different run unit, except by the execution of a READ statement with the IGNORING LOCK phrase. A locked record may be re-accessed by the same file connector that holds the lock.

In all cases, all record locks established for a file are released by the execution of an explicit or implicit CLOSE statement for the file.

The implementor may specify circumstances other than a locked logical record that result in the return of a locked record status.

NOTE Examples of such circumstances are the locking of records while an index is being organized and the locking of a physical block containing a locked logical record. By defining the physical file to contain one logical record per physical record, users can avoid the situation where a record locked status occurs for a record because it is contained in a block in which a different record is locked.

9.1.16 Sort file

A sort file is a collection of records to be sorted by a SORT statement. The rules for blocking and for allocation of internal storage are peculiar to the SORT statement. The RELEASE and RETURN statements imply nothing with respect to buffer areas, blocks, or reels. A sort file, then, can be considered as an internal file that is created (RELEASE statement) from the input file, processed (SORT statement), and then made available (RETURN statement) to the output file.

A sort file is named by a file control entry and is described by a sort-merge file description entry. The only statements that may reference a sort file are the RELEASE, RETURN, and SORT statements.

9.1.17 Merge file

A merge file is a collection of records to be merged by a MERGE statement. The rules for blocking and for allocation of internal storage are peculiar to the MERGE statement. The RETURN statement implies nothing with respect to buffer areas, blocks, or reels. A merge file, then, can be considered as an internal file that is created from input files by combining them (MERGE statement) as the file is made available (RETURN statement) to the output file.

A merge file is named by a file control entry and is described by a sort-merge file description entry. The only statements that may reference a merge file are the RETURN and MERGE statements.

9.1.18 Dynamic file assignment

Dynamic file assignment allows the user to defer until runtime the association between a file connector and a physical file. This feature may also be used to associate one file connector with different physical files during execution of a run unit. It is specified by the USING phrase of the ASSIGN clause on the file control entry. The USING phrase references an alphanumeric data item whose content at the time an OPEN, SORT, or MERGE statement for that file is executed uniquely identifies the specific physical file to be accessed.

9.1.19 Report file

A report file is an output file having sequential organization whose file description entry contains a REPORT clause.

9.2 Screens

9.2.1 Terminal screen

A terminal provides I-O via a screen and a keyboard. A screen is considered a grid of rows and columns, where the size of a column is one fixed-size alphanumeric character position. There is a one-to-one correspondence between a column and a character in the computer's alphanumeric coded character set. There is a fixed correspondence, specified by the implementor, between a column and a character in the computer's national coded character set.

NOTE 1 This final committee draft International Standard does not specify the manner of presenting data on a screen in a proportional font.

A screen contains one or more fields during each input or output operation. A field may range in size from one character to the maximum number of characters permitted on the screen. Each field represents an elementary screen item. One or more fields may be logically grouped together into a group screen item; such fields need not be contiguous. A group screen item may contain other group screen items. The fields within a group screen item are ordered for the purposes of determining the next field and the previous field operations during terminal input. The order of fields is determined by the order of declaration of screen items in a screen description entry.

NOTE 2 A group screen item, unlike a group data item, is never treated as a single contiguous string of characters of a particular category.

A screen has visible attributes associated with each display location.

9.2.2 Function keys

A function key has a function key number associated with it that is returned to the run unit when it is pressed.

The implementor may define context-dependent function keys to carry out a particular function in a specific context. If any context-dependent function key is defined, the implementor shall specify the function number that is returned for each context-dependent function key.

The implementor shall specify the method, if one exists, for enabling and disabling function keys and context-dependent function keys.

9.2.3 CRT status

The CRT status is a four-character conceptual entity whose value is set to indicate the status of a terminal input-output operation during the execution of an ACCEPT screen statement and prior to the execution of any imperative statement associated with any ON EXCEPTION or NOT ON EXCEPTION clauses for that ACCEPT statement. The value of the CRT status is available through the use of the CRT STATUS clause in the SPECIAL-NAMES paragraph.

CRT status expresses one of the following conditions upon completion of the input operation:

1) Successful completion with normal termination. The input statement was successfully executed.

2) Successful completion with termination by a function key keystroke. The input statement was successfully executed.

3) Unsuccessful completion. The input statement was not successfully executed. Further terminal I-O statements are not precluded.

4) Implementor-defined unsuccessful completion.

The following is a list of the values placed in the CRT status for the conditions resulting from the execution of an input operation.

1) Successful completion with normal termination

   CRT status = 0000. The input statement was successfully executed. Termination was achieved by the operator pressing the enter key or entering data into the last character of a screen item for which the AUTO clause is specified and for which no logical next field exists.

2) Successful completion with termination by a function key keystroke

   a) CRT status = 1xxx. The input statement was successfully executed. Termination was achieved by the operator pressing a function key. The number of the function key that was pressed is given by the numeric value of xxx.

   b) CRT status = 2xxx. The input statement was successfully executed. Termination was achieved by the operator pressing a context-dependent function key. The number of the function assigned to the key that was pressed is given by the numeric value of xxx.

3) Unsuccessful completion with standard-defined condition

   a) CRT status = 8000. The ACCEPT screen statement was unsuccessful because no input screen item was located at a valid screen position.

   b) CRT status = 8001. The ACCEPT screen statement was unsuccessful because inconsistent data was entered into a screen item and allowed to remain there.

      NOTE On some implementations, this will not occur because the operator is forced to correct the data before execution will continue.

4) Unsuccessful completion with implementor-defined condition

   a) CRT status = 9xxx. An implementor-defined condition exists. The numeric value xxx is defined by the implementor.

9.2.4 Cursor

Character addressable terminals use the concept of a cursor to indicate the position on the screen at which keyboard operations will be displayed. This is generally indicated by the position of a visible cursor symbol.

During execution of a DISPLAY screen statement, the position and visibility of the cursor is undefined.

During execution of an ACCEPT screen statement, the position and visibility of the cursor is defined only during the period that the keyboard is synchronously enabled for operator input; the cursor shall be visible and shall indicate the position on the screen at which keyboard input will be displayed.

During execution of an ACCEPT screen statement, the cursor is initially positioned at the first elementary screen item in the screen description entry whose specification includes a TO or USING phrase, unless the CURSOR clause is specified in the SPECIAL-NAMES paragraph, in which case the cursor is positioned as specified in that clause.

Once the keyboard is enabled for operator input, the operator may move the cursor to elementary screen items whose specification includes a TO or USING clause. Depending on the screen description entry for the item, the operator may move the cursor to characters within the displayed item.

The implementor shall specify any keys that change the position of the cursor and the associated cursor movement.

9.2.5 Cursor locator

The cursor locator is a six-character conceptual entity whose value is set by the runtime element to indicate the position of the visible cursor on the display screen when the keyboard becomes synchronously enabled during execution of an ACCEPT screen statement. The position is relative to the top left hand corner of the screen.

Upon successful termination of execution of an ACCEPT screen statement, the cursor locator is set to indicate the position of the visible cursor at the time the operator presses the terminator key or a function key. If the execution of the ACCEPT statement was unsuccessful, the value of the cursor locator is undefined.

The cursor locator is made available to the runtime element through the use of the CURSOR clause in the SPECIAL-NAMES paragraph. The first three characters represent a three-digit number giving the line number, the topmost line being 001. The second three characters represent a three-digit number giving the column number, the first column number being 001. If the position of the visible cursor is at a line or column number that is greater than 999, the value of the cursor locator is undefined.

9.2.6 Current screen item

During the execution of an ACCEPT statement, one or more elementary input screen items can be displayed on the terminal display. The operator is able to move the cursor between the screen items using context-dependent cursor positioning keys. The cursor may also move automatically from one screen item to another when the screen item becomes full or the last character in the screen item is keyed. The screen item in which the cursor is located is the current screen item. Any data keyed by the operator is assigned to the current screen item and might cause the display of the current screen item to change.

9.2.7 Color number

Color is one of the attributes that may be specified for screen items. For a monochrome terminal, the color attributes are mapped onto other attributes by the implementor.

A color is selected by specifying an integer that represents the color. The colors and their associated color numbers are:

black	0
blue	1
green	2
cyan	3
red	4
magenta	5
brown/yellow	6
white	7

NOTE The colors above are a rough guide; the actual color depends on the terminal capabilities and may be affected by other considerations such as the HIGHLIGHT attribute. For example, the value 6 might appear as brown, but when HIGHLIGHT is also specified it might appear as yellow. The value 0 might appear as black, but when HIGHLIGHT is also specified it might appear as gray.

9.3 Objects

9.3.1 Objects and classes

An object is an information processing unit consisting of data and methods. Methods are units of code designed to act on the data of objects. Each object contains its own instance of data and file connectors and shares the methods that are defined for that object with other objects of its class.

A class is the template from which objects are made. The source unit that defines the template is a class definition, which specifies the characteristics of data and the methods of an object. A class may describe a factory object and an instance object. There is at most one runtime instance of a factory object for a given class in a given run unit. There may be any number of instances of an instance object in a given run unit at any given time.

9.3.2 Object references

An object reference is an implicitly- or explicitly-defined data item containing an object reference value that uniquely references an object for the lifetime of the object. Implicitly-defined object references are the predefined object references and object references returned from an object property, an object-view, an inline method invocation, or a function. Explicitly-defined object references are data items defined by a data description entry specifying a USAGE OBJECT REFERENCE clause.

No two distinct objects have the same object reference value and every object has at least one object reference.

9.3.3 Predefined object references

A predefined object reference is an implicitly-generated data item referenced by one of the identifiers EXCEPTION-OBJECT, NULL, SELF, and SUPER. Each predefined object reference has a specific meaning, as described in 8.4.2, Identifiers.

9.3.4 Methods

The procedural code in an object is placed in methods. Each method has its own method-name and its own data division and procedure division. When a method is invoked, the procedural code it contains is executed. A method is invoked by specifying an identifier that references the object and the name of the method. A method may specify parameters and a returning item. A method always possesses the recursive attribute and may call itself.

9.3.5 Polymorphism

Polymorphism is a feature of object orientation that allows a given method to do different things. COBOL supports three types of polymorphism: class polymorphism, ad-hoc polymorphism and parametric polymorphism.

9.3.5.1 Class polymorphism

In COBOL, the ability for an object reference to contain references to objects of different classes allows a method invocation on that object reference to be bound to one of many possible methods. Sometimes the method can be identified before execution, but in general, the method cannot be identified until runtime.

A data item can be declared to contain references to objects of a given class or any sub-class of that class; a data item can also be declared to contain references to objects that implement a given interface. When a given interface is used, the classes of the objects may be completely unrelated as long as they implement the given interface.

9.3.5.2 Parametric polymorphism

Parametric polymorphism, sometimes called method overlaoding, allows methods within a class to have the same method name, differing in the number and type of parameters. The most appropriate method for any particular invocation is chosen during method resolution. Each method has a method resolution signature that is matched against the parameters of the invocation. The method resolution signature contains all of the information necessary to resolve the identity of the method invoked. The method resolution signature consists of:

1) The method name

2) The expected calling convention

3) The DECIMAL-POINT IS COMMA clause

4) The CURRENCY SIGN clause

5) The LOCALE clause

6) A description of each parameter, and of the returning item. This description consists of the following clauses:

    a) ANY-LENGTH clause

    b) BLANK WHEN ZERO clause

    c) JUSTIFIED clause

    d) PICTURE clause

    e) SIGN clause

    f) USAGE clause

Parametric polymorphism is an optional feature in this final committee draft International Standard. When parametric polymorphism is provided, the manner of method resolution may be as defined in 9.3.6, Method invocation, or may be based on a methodology used in the processor's object management system.

9.3.6 Method invocation

The procedural code in a method is executed by invoking the method either with an INVOKE statement, an inline method invocation, or a reference to an object property. The method implementation that is bound to the invocation depends on the class, at runtime, of the object on which the method is invoked. In particular, it is not necessarily the class specified statically in the definition of the object reference; it is the class of the actual object referenced at runtime that is used in resolving a method invocation to a particular method implementation.

If an invocation specifies the object using an object reference, then:

1) if the identified object is a factory object, the method invocation will resolve to a factory method

2) otherwise, the resolution will be to an instance method.

Additionally, if an invocation specifies the object using SUPER, the invocation will resolve to a method by using a restricted search, as specified in 8.4.2.8, SELF and SUPER.

If an invocation specifies the object using a class name, the factory object of the specified class is used as the object on which the method is invoked, and the method invocation will resolve to a factory method.

Method resolution proceeds as follows:

1) if a method with the method-name specified in the invocation is defined in the class of the object and that method is an exact match between the method invocation and the method resolution signature as specified below, that method is bound;

2) otherwise, each inherited class upward in the hierarchy of inheritance is inspected in the order in which the hierarchy is specified in the INHERITS clause until either a method with the method-name specified in the invocation is defined in the inspected class and is an exact match, or all inherited classes have been inspected without finding such a method. If a method is found, that method is bound;

3) otherwise, if one or more methods with the method-name specified in the invocation is defined in the class of the object and one or more of those methods match between the method invocation and the method resolution signature, that method is bound as specified below;

4) otherwise, each inherited class upward in the hierarchy of inheritance is inspected in the order in which the hierarchy is specified in the INHERITS clause until either one or more methods with the method-name specified in the invocation is defined in the inspected class and one or more of those methods match between the method invocation and the method resolution signature, or all inherited classes have been inspected without finding such a method. If a method is found, that method is bound;

5) otherwise, if any invocation parameter is a literal, steps 3 and 4 are repeated, ignoring the requirements specified in 4c, 4d, 5c and 5d below that the SET and MOVE statements not cause truncation;

6) otherwise, the EC-OO-METHOD exception condition is set to exist.

For a method invocation to match the method resolution signature of a method, the following shall be true:

1) The number of invocation parameters shall be equal to the number of parameters defined in the invoked method. If the number of parameters in the invocation is less than the number of parameters in the invoked method, and all parameters to the right of the parameter in the position that corresponds to the number of parameters in the invocation are described with the OPTIONAL phrase, the invocation and the invoked method shall be considered to have an equal number of parameters. If there is a returning item in the invocation there shall be a returning item in the invoked method, if there is no returning item in the invocation there shall be no returning item in the invoked method.

2) The same calling convention shall be in effect for the invocation as is in effect for the invoked method.

3) For each parameter of the invocation that is passed by reference there shall be a corresponding parameter in the invoked method that:

   a) is declared with the BY REFERENCE phrase

   b) if the passed parameter is declared with the OMITTED phrase, shall be declared with the OPTIONAL phrase. No further checking is performed on this parameter and this parameter is considered to match exactly.

   c) is the same class and category

   d) if the parameter in the invoked method is an object reference, the corresponding parameter shall follow these rules:

      1. If the parameter in the invoked method is a universal object reference, the corresponding parameter in the invocation is a universal object reference.

      2. If the parameter in the invoked method is described with an interface-name, the corresponding parameter is described with the same interface- name.

      3. If the parameter in the invoked method is described with a class-name, the corresponding parameter in the invocation is described with the same class-name, and the presence or absence of the FACTORY and ONLY phrases is the same in both descriptions.

      4. If the parameter in the invoked method is described with the ACTIVE-CLASS phrase and FACTORY phrase is not specified, the corresponding parameter shall evaluate to an object reference of the same class specified in the invocation.

      5. If the parameter in the invoked method is described with the ACTIVE-CLASS and FACTORY phrases, the corresponding parameter shall evaluate to an object reference to the factory of the class specified in the invocation.

e) If the parameter in the invoked method is not an object reference, the corresponding parameter has the same ANY LENGTH, BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, and USAGE clauses, with the following additional constraints:

1. Currency symbols match if and only if the corresponding currency strings are the same.

2. Period picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these interfaces. Comma picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these interfaces.

3. Locale specifications in the PICTURE clauses match if and only if:

a. both specify the same SIZE phrase in the LOCALE phrase of the PICTURE clause, and

b. both specify the LOCALE phrase without a locale-name or both specify the LOCALE phrase with the same external identification, where the external identification is the external-locale-name or literal value associated with the locale-name in the LOCALE clause of the SPECIAL-NAMES paragraph.

4) For each parameter that is passed by content there shall be a corresponding parameter in the invoked method that:

a) is declared with the BY REFERENCE phrase

b) if the passed parameter is declared with the OMITTED phrase, shall be declared with the OPTIONAL phrase. No further checking is performed on this parameter and this parameter is considered to match exactly.

c) if the parameter in the invoked method is class OBJECT, POINTER, or INDEX, may be a receiving data item in a SET statement with the parameter of the invocation as a sending data item. If the sending data item is a literal, the SET statement shall not cause truncation.

d) if the parameter in the invoked method is not class OBJECT, POINTER, or INDEX, may be a receiving data item in a MOVE statement with the parameter of the invocation as a sending data item. If the sending data item is a literal, the MOVE statement shall not cause truncation.

5) For each parameter that is passed by value there shall be a corresponding parameter in the invoked method that:

a) is declared with the BY VALUE phrase

b) if the passed parameter is declared with the OMITTED phrase, shall be declared with the OPTIONAL phrase. No further checking is performed on this parameter and this parameter is considered to match exactly.

c) if the parameter in the invoked method is class OBJECT, POINTER, or INDEX, may be a receiving data item in a SET statement with the parameter of the invocation as a sending data item. If the sending data item is a literal, the SET statement shall not cause truncation.

d) if the parameter in the invoked method is not class OBJECT, POINTER, or INDEX, may be a receiving data item in a MOVE statement with the parameter of the invocation as a sending data item. If the sending data item is a literal, the MOVE statement shall not cause truncation.

6) If the invocation specifies a returning item and the returning item is usage OBJECT REFERENCE, POINTER or INDEX, there shall be a corresponding specification in the invoked method that may be a sending item in a SET statement with the returning item as the receiving item.

7) If the invocation specifies a returning item and the returning item is not usage OBJECT REFERENCE, POINTER or INDEX, there shall be a corresponding specification in the invoked method that may be a sending item in a MOVE statement with the returning item as the receiving item.

Additionally, for each parameter that is passed by value or by content, and for the returning item, the following criteria are considered an exact match:

1) the parameter or returning item in the invocation and the corresponding parameter or returning item in the invoked method are the same class and category.

2) if the parameter or returning item in the invoked method is an object reference, the corresponding parameter or returning item shall follow these rules:

    a) If the parameter or returning item in the invoked method is a universal object reference, the corresponding parameter or returning item in the invocation is a universal object reference.

    b) If the parameter or returning item in the invoked method is described with an interface-name, the corresponding parameter or returning item in is described with the same interface- name.

    c) If the parameter or returning item in the invoked method is described with a class-name, the corresponding parameter or returning item in the invocation is described with the same class-name, and the presence or absence of the FACTORY and ONLY phrases is the same in both descriptions.

    d) If the parameter in the invoked method is described with the ACTIVE-CLASS phrase and FACTORY phrase is not specified, the corresponding parameter shall evaluate to an object reference of the same class specified in the invocation.

    e) If the parameter in the invoked method is described with the ACTIVE-CLASS and FACTORY phrases, the corresponding parameter shall evaluate to an object reference to the factory of the class specified in the invocation.

3) if the parameter or returning item in the invoked method is not an object reference, the corresponding parameter or returning item has the same ANY LENGTH, BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, and USAGE clauses, with the following additional constraints:

    a) Currency symbols match if and only if the corresponding currency strings are the same.

    b) Period picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these interfaces. Comma picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these interfaces.

    c) Locale specifications in the PICTURE clauses match if and only if:

        1. both specify the same SIZE phrase in the LOCALE phrase of the PICTURE clause, and

        2. both specify the LOCALE phrase without a locale-name or both specify the LOCALE phrase with the same external identification, where the external identification is the external-locale-name or literal value associated with the locale-name in the LOCALE clause of the SPECIAL-NAMES paragraph.

For the purpose of determining an exact match, an alphanumeric, boolean, or national literal is considered to be equivalent to a data item with no PICTURE clause and a VALUE clause that specifies that literal. If the literal is a numeric literal: the implied usage is DISPLAY; the implied picture clause consists of the same number of picture symbol characters '9' as there are digits in the literal; the presence or absence of signs is the same; the decimal separator is the same; and the decimal separator is in the same position within the picture character-string as in the literal.

All parameters passed by reference are exact matches. A method invocation is considered to match exactly if all parameters match exactly.

If two or more methods in a class match, preference is given to the method that has the greatest number of parameters that match exactly. If the number of parameters that match exactly are equal, preference is given first to the methods with the least number of exact matches due to omitted optional parameters, second to those methods where the returning item is an exact match, and then to the first method defined in the class or interface.

9.3.7 Method prototypes

Method definitions within an interface definition define method prototypes. Method prototypes do not specify procedural code, but rather they specify the details needed to interface with and check the conformance of a method.

9.3.8 Conformance and interfaces

The term 'conformance' is used in this document with several different meanings. In the context of object orientation, the term 'conformance' is used to describe a relationship between object interfaces, and it is the basis of such fundamental features as inheritance, interface definitions, and conformance checking.

NOTE Conformance checking is done at compile time only, except that conformance checking for object views and methods using universal object references is done at runtime.

9.3.8.1 Conformance for object orientation

Conformance for objects allows an object to be used according to an interface other than the interface of its own class. Conformance is a unidirectional relation from one interface to another interface and from an object to an interface.

9.3.8.1.1 Interfaces

Every object has an interface consisting of the names and parameter specifications for each method supported by the object, including inherited methods. Each class has two interfaces: an interface for the factory object and an interface for the instance objects.

Interfaces may also be defined independently from a specific class by specifying method prototypes in an interface definition.

9.3.8.1.2 Conformance between interfaces

If an interface interface-1 and an interface interface-2 are the same interface, they conform to each other. If interface-1 and interface-2 are different interfaces, interface-1 conforms to interface-2 if and only if the entry conventions for interface-1 and interface-2 are the same and for every method in interface-2 there is a method in interface-1 with the same name that satisfies the following conditions:

1) The number of parameters are the same, with consistent BY REFERENCE and BY VALUE specifications.

2) If the formal parameter of the method in interface-2 is an object reference, the corresponding parameter in interface-1 is an object reference following these rules:

    a) If the parameter in interface-2 is a universal object reference, the corresponding parameter in interface-1 is a universal object reference.

    b) If the parameter in interface-2 is described with an interface-name, the corresponding parameter in interface-1 is described with the same interface- name.

    c) If the parameter in interface-2 is described with a class-name, the corresponding parameter in interface-1 is described with the same class-name, and the presence or absence of the FACTORY and ONLY phrases is the same in both interfaces.

d) If the parameter in interface-2 is described with the ACTIVE-CLASS phrase, the corresponding parameter in interface-1 is described with the ACTIVE-CLASS phrase, and the presence or absence of the FACTORY phrase is the same in both interfaces.

3) If the formal parameter of the method in interface-2 is not an object reference, the corresponding formal parameter in interface-1 has the same ANY LENGTH, BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, and USAGE clauses, with the following exceptions:

a) Currency symbols match if and only if the corresponding currency strings are the same.

b) Period picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these interfaces. Comma picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these interfaces.

Additionally, locale specifications in the PICTURE clauses match if and only if:

— both specify the same SIZE phrase in the LOCALE phrase of the PICTURE clause, and

— both specify the LOCALE phrase without a locale-name or both specify the LOCALE phrase with the same external identification, where the external identification is the external-locale-name or literal value associated with the locale-name in the LOCALE clause of the SPECIAL-NAMES paragraph.

4) The presence or absence of the procedure division RETURNING phrase is the same.

5) If the returning item of the method of interface-2 is an object reference, the corresponding returning item in interface-1 is an object reference following these rules:

a) If the returning item in interface-2 is a universal object reference, the corresponding returning item in interface-1 is an object reference.

b) If the returning item in interface-2 is described with an interface-name that identifies the interface int-r, the corresponding returning item in interface-1 is either of the following:

1. an object reference described with an interface-name that identifies int-r or an interface described with an INHERITS clause that references int-r,

2. an object reference described with a class-name, subject to the following rules:

a. if described with the FACTORY phrase, the factory object of the specified class shall be described with an IMPLEMENTS clause that references int-r,

b. if described without the FACTORY phrase, the instance objects of the specified class shall be described with an IMPLEMENTS clause that references int-r.

c) If the returning item in interface-2 is described with a class-name, the corresponding returning item in interface-1 is an object reference, subject to the following rules:

1. If the returning item in interface-2 is described with the ONLY phrase, the returning item in interface-1 shall be described with the ONLY phrase and the same class-name.

2. If the returning item in interface-2 is described without the ONLY phrase, the returning item in interface-1 shall be described with the same class-name or a subclass of that class-name.

3. The presence or absence of the FACTORY phrase shall be the same.

d) If the returning item in interface-2 is described with the ACTIVE-CLASS phrase, the corresponding returning item in interface-1 is described with the ACTIVE-CLASS phrase, and the presence or absence of the FACTORY phrase is the same.

If the description of the returning item of a method in interface-1 directly or indirectly references interface-2, the description of the returning item of the corresponding method in interface-2 shall not directly or indirectly reference interface-1.

6) If the returning item of the method of interface-2 is not an object reference, the corresponding returning item has the same ANY LENGTH, BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, and USAGE clauses, with the following exceptions:

   a) Currency symbols match if and only if the corresponding currency strings are the same.

   b) Period picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these interfaces. Comma picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both or for neither of these interfaces.

   Additionally, locale specifications in the PICTURE clauses match if and only if:

   — both specify the same SIZE phrase in the LOCALE phrase of the PICTURE clause, and

   — both specify the LOCALE phrase without a locale-name or both specify the LOCALE phrase with the same external identification, where the external identification is the external-locale-name or literal value associated with a locale-name in the LOCALE clause of the SPECIAL-NAMES paragraph.

7) If either of the corresponding formal parameters or returning items in interface-1 or interface-2 is a strongly-typed group item, both are of the same type.

8) The presence or absence of the OPTIONAL phrase is the same for corresponding parameters.

9) If the RAISING phrase is specified in the procedure division header of the method in interface-1, the corresponding method in interface-2 specifies the RAISING phrase following these rules:

   a) If an exception-name is specified in the RAISING phrase in interface-1, the corresponding RAISING phrase in interface-2 specifies the same exception-name.

   b) If a class-name is specified in the RAISING phrase in interface-1, the corresponding RAISING phrase in interface-2 specifies one of the following:

      — the same class-name or the name of a superclass of the class identified by that class-name, including the FACTORY phrase if and only if the RAISING phrase in interface-1 specifies the FACTORY phrase,

      — the name of an interface implemented by the factory object of that class, if the RAISING phrase in interface-1 specifies the FACTORY phrase,

      — the name of an interface implemented by the instance object of that class, if the RAISING phrase in interface-1 does not specify the FACTORY phrase.

   c) If an interface-name is specified in the RAISING phrase in interface-1, the corresponding RAISING phrase in interface-2 specifies the same interface-name or the name of an interface inherited by that interface.

An alphanumeric group item that is not strongly typed is, for the purpose of conformance checking, considered to be equivalent to an elementary alphanumeric data item of the same length.

Variable-length groups conform if they are compatible groups, and all variable-length data items correspond to variable-length data items as specified in 8.5.1.9, Variable-length groups.

9.3.8.1.3 Conformance for parameterized classes and parameterized interfaces

When using a parameterized class or interface, the class or interface is treated as if the actual parameter classes or interfaces were substituted for the parameters throughout the class definition or interface definition.

9.3.9 Class inheritance

Class inheritance is a mechanism for using the interface and implementation of one or more classes as the basis for another class. The inheriting class, also known as a subclass, inherits from one or more classes, known as superclasses. The subclass has all the methods defined for the inherited class definition or definitions, including any methods that the inherited definition or definitions inherited. The subclass has all the data definitions defined for the inherited class or classes, including any data definitions that the inherited class or classes inherited.

NOTE This does not mean that the actual source code that describes the data is accessible or that the data items described in that source code can be directly referenced in the subclass. It means that the subclasses are treated as if their source code had a copy of the superclass definitions; in other words, the inherited data items are considered to be defined in the subclass.

The inherited data definitions define data for every instance object of the subclass and for its factory object. Each instance object has its own copy of inherited data, distinct from the copy belonging to an instance object of an inherited class. Each factory object has its own copy of inherited data, distinct from the copy belonging to a factory object of an inherited class. Names and attributes of inherited data items are not visible in the inheriting class. The inherited object data is initialized when an object is created. The inherited factory data is allocated independently from the factory data of the inherited class or classes and is initialized when the factory of the subclass is created. The inherited factory data is accessible only via methods and properties specified in the factory definition of the class that describes the data. The inherited object data is accessible only via methods and properties specified in the object definition of the class that describes the data. The subclass inherits all the file definitions in the same way as the data definitions, subject to the same provisions as data definitions. The subclass may define methods in addition to or in place of the inherited methods and may specify data definitions and file definitions in addition to, but not in place of, the inherited data definitions and file definitions.

The interface of a subclass shall always conform to the interface of the inherited classes, although the subclass may override some of the methods of the inherited class to provide different implementations.

If a class is defined with the FINAL clause, that class shall not be used as a superclass. If a method is defined with the FINAL clause, that method shall not be overridden in a subclass.

User-defined words in an inherited class are not visible in the subclass. Any such word may be used as any type of user-defined word in the subclass definition.

9.3.10 Interface inheritance

Interface inheritance is a mechanism for using one or more interface definitions as the basis for another interface. The inheriting interface has all the method specifications defined for the inherited interface definition or definitions, including any method specifications that the inherited definition or definitions inherited. The inheriting interface may define new methods augmenting the set of inherited method specifications. The inheriting interface shall always conform to each of the inherited interfaces.

9.3.11 Interface implementation

Interface implementation is a mechanism for using one or more interface definitions as the basis for a class. The implementing class shall implement all the method specifications defined for the implemented interface definition or definitions, including any method specifications that the implemented definition or definitions inherited. The interface of the factory object of the implementing class shall conform to the interfaces to be implemented by the factory object, and the interface of the instance objects of the implementing class shall conform to the interfaces to be implemented by the instance object.

9.3.12 Parameterized classes

A parameterized class is a generic or skeleton class that has formal parameters that will be replaced by one or more class-names or interface-names. When it is expanded by substituting specific class-names or interface-names as actual parameters, a class is created that functions as a non-parameterized class.

An expansion of a parameterized class is treated in all respects the same as if it were a class that is not a parameterized class.

When a class is specified as the parameterized class in an EXPANDS phrase in the REPOSITORY paragraph, a new class (an instance of a parameterized class) is created based on the specification of the parameterized class. This class has its own factory object and is completely separate from any other instance of the same parameterized class.

Within a run unit, two classes with the same externalized class-name that are created by expanding the same parameterized class with the same actual parameters are the same class instance. If two classes expand a parameterized class with different actual parameters, they are not the same class instance and shall not have the same externalized class-name.

9.3.13 Parameterized interfaces

A parameterized interface is a generic or skeleton interface that has formal parameters that will be replaced by one or more class-names or interface-names. When it is expanded by substituting specific class-names or interface-names as actual parameters, an interface is created that functions as a non-parameterized interface.

An expansion of a parameterized interface is treated in all respects as if it were an interface that is not parameterized.

When an interface is specified as the parameterized interface in an EXPANDS phrase in the REPOSITORY paragraph, a new interface (an instance of a parameterized interface) is created based on the specification of the parameterized interface.

Within a run unit, two interfaces with the same externalized interface-name that are created by expanding the same parameterized interface with the same actual parameters are the same interface instance. If two interfaces expand a parameterized interface with different actual parameters, they are not the same interface instance and shall not have the same externalized interface-name.

9.3.14 Object life cycle

The life cycle for an object begins when it is created and ends when it is destroyed.

9.3.14.1 Life cycle for factory objects

A factory object is created before it is first referenced by a run unit.

A factory object is destroyed after it is last referenced by a run unit.

9.3.14.2 Life cycle for instance objects

An instance object is created as the result of the NEW method being invoked on a factory object.

An instance object is destroyed either when it is determined that the object cannot take part in the continued execution of the run unit, or when the run unit terminates, whichever occurs first.

The timing and algorithm for the mechanism that determines whether or not an instance object can take part in the continued execution of the run unit is implementor-defined.

NOTE The process of determining whether or not an instance object can take part in continued execution and reclaiming resources unique to the object is generally referred to as garbage collection.

9.4 User-defined functions

A user-defined function is an entity that is defined by the user by specifying a FUNCTION-ID paragraph rather than a PROGRAM-ID paragraph. The rules and behavior of a user-defined function are similar to those for a program except that a user-defined function returns a value as specified by the RETURNING phrase in the procedure division header. Also, arguments and returned values for user-defined functions cannot use the word ALL as a subscript. In

addition, a user defined function always possesses the recursive attribute and may call itself. A user-defined function is invoked by specifying a function identifier as described in 8.4.2.2, Function-identifier.

10 Structured compilation group

10.1 General

A structured compilation group consists of zero, one, or more compilation units that have been processed by text manipulation. A structured compilation group may contain compiler directives affecting compilation processing or source listings, as specified in 7, Compiler directing facility. Text manipulation compiler directives may be logically represented in a structured compilation group but have no effect on compilation processing.

10.2 Compilation units

A compilation unit is one of the following:

- a program-prototype definition
- a function-prototype definition
- a program definition for an outermost program, including its nested programs
- a class definition
- an interface definition
- a function definition

Successful compilation of each compilation unit that is a program definition, a function definition, or a class definition results in executable code that, when included in a run unit, constitutes a runtime module.

A compilation unit may contain one or more source units, depending on the type of definition.

The compilation units in a compilation group may be all or part of a run unit or may be unrelated compilation units.

10.3 Source units

A source unit begins with an identification division and ends with an end marker or the end of the compilation group. A source unit includes any contained source units. The following are source units:

- an outermost program-definition, including its contained program-definitions
- a contained program-definition, including its contained program-definitions
- a program-prototype-definition
- a function-definition
- a function-prototype-definition
- a class-definition, including its factory definition and instance definition
- a factory definition, including its method definitions
- an instance definition, including its method definitions
- a method definition
- an interface definition, including its method prototypes

A source unit may contain one or more divisions, specified in the following order:

1) identification division
2) environment division
3) data division
4) procedure division

The beginning of a division is indicated by its division header or, in the case of the identification division when its header is omitted, by one of the paragraph headers permitted in the identification division.

The end of a division is indicated by the beginning of the next division, an end marker for the source unit, or the end of the compilation group.

10.4 Contained source units

Source units may be contained directly or indirectly.

The factory definition and instance definition in a class definition are directly contained within that class definition. The methods within a factory definition or an instance definition are directly contained within that factory definition or instance definition, respectively.

Program-definitions within another program-definition may be contained directly or indirectly. A program definition directly contains another program-definition that is immediately nested within it. A program-definition indirectly contains another program-definition when there is one or more levels of nesting between the two of them.

When source units are contained within other source units, names of resources described in the containing source unit may be referenced in the contained source unit in accordance with the rules specified in 8.4.5, Scope of names.

The executable code resulting from compilation of a contained source unit is considered inseparable from the executable code resulting from compilation of the containing source unit.

10.5 Source elements and runtime elements

A source element is a source unit excluding any contained source units.

NOTE — In the example:

```
PROGRAM-ID.  A.
    ...
    PROGRAM-ID.  B.
    ...
        PROGRAM-ID.  C.
        ...
        END PROGRAM C.
    END PROGRAM B.
END PROGRAM A.
```

Program B is directly contained in Program A; Program C is directly contained in Program B; and Program C is indirectly contained in Program A. Program C is a source element; Program B, devoid of Program C, is a source element; and Program A, devoid of programs B and C, is a source element.

A runtime element is the result of successful compilation of a function, a method, or a program containing a procedure division and consists of executable code included in a run unit.

10.6 COBOL compilation group

10.6.1 General format

```
[ { program-prototype     } ... ]
  { function-prototype    }
  { program-definition    }
  { function-definition   }
  { class-definition      }
  { interface-definition  }
```

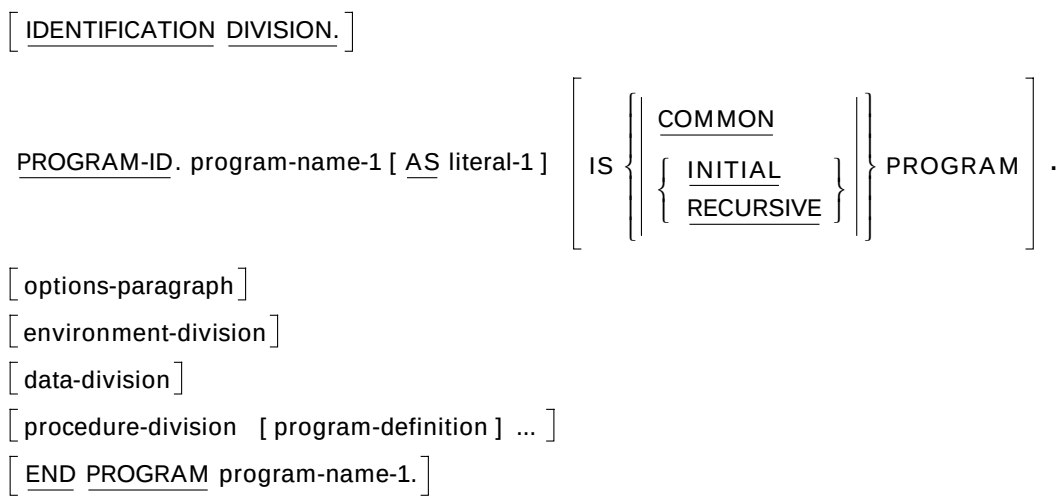
where program-prototype is:

```
[ IDENTIFICATION DIVISION. ]
PROGRAM-ID. program-prototype-name-1 [ AS literal-1 ] IS PROTOTYPE.
[ options-paragraph ]
[ environment-division ]
[ data-division ]
[ procedure-division ]
END PROGRAM program-prototype-name-1.
```

where function-prototype is:

```
[ IDENTIFICATION DIVISION. ]
FUNCTION-ID. function-prototype-name-1 [ AS literal-1 ] IS PROTOTYPE.
[ options-paragraph ]
[ environment-division ]
[ data-division ]
[ procedure-division ]
END FUNCTION function-prototype-name-1.
```

where program-definition is:



where function-definition is:

```
[ IDENTIFICATION DIVISION. ]
FUNCTION-ID.  user-function-name-1 [ AS literal-1 ].
[ options-paragraph ]
[ environment-division ]
[ data-division ]
[ procedure-division ]
END FUNCTION  user-function-name-1.
```

where class-definition is:

```
[ IDENTIFICATION DIVISION. ]
CLASS-ID. class-name-1 [ AS literal-1 ] [ IS FINAL ]
    [ INHERITS FROM { class-name-2 }  ...  ]
    [ USING { parameter-name-1 }  ...  ]  .
 [ options-paragraph ]
 [ environment-division ]
 [ factory-definition ]
 [ instance-definition ]
 END CLASS class-name-1.
```

where factory-definition is:

```
[ IDENTIFICATION DIVISION. ]
FACTORY. [ IMPLEMENTS { interface-name-1 } ... . ]
[ options-paragraph ]
[ environment-division ]
[ data-division ]
[ procedure-division ]
END FACTORY.
```

where instance-definition is:

```
[ IDENTIFICATION DIVISION. ]
OBJECT. [ IMPLEMENTS { interface-name-2 } ... . ]
[ options-paragraph ]
[ environment-division ]
[ data-division ]
[ procedure-division ]
END OBJECT.
```

where interface-definition is:

```
[ IDENTIFICATION DIVISION. ]
INTERFACE-ID.  interface-name-1 [ AS literal-1 ]
    [ INHERITS FROM { interface-name-2 } ... ]
    [ USING { parameter-name-1 } ... ] .
[ options-paragraph ]
[ environment-division ]
[ procedure-division ]
END INTERFACE interface-name-1.
```

where method-definition is:

```
[ IDENTIFICATION DIVISION. ]
METHOD-ID. { method-name-1 [ AS literal-1 ]
           { { GET } }
           { { SET } } PROPERTY property-name-1 } [ OVERRIDE ] [ IS FINAL ] .
[ options-paragraph ]
[ environment-division ]
[ data-division ]
[ procedure-division ]
END METHOD [ method-name-1 ] .
```

NOTE Method-definition is included here for completeness, because it is a type of source element. Method-definition is referenced in the general format of 14, Procedure division. A method-definition in a class-definition defines a method. A method-definition in an interface-definition defines a method prototype.

where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
data-division	13, Data division
environment-division	12, Environment division
options-paragraph	11.9, OPTIONS paragraph
procedure-division	14, Procedure division

10.6.2 Syntax rules

1) Within a compilation group, function-prototypes and program-prototypes shall precede all other types of source units.

2) If a compilation group contains both a program definition and a program prototype definition with the same externalized name, the signatures of these two compilation units shall be the same.

3) If a compilation group contains both a function definition and a function prototype definition with the same externalized name, the signatures of these two compilation units shall be the same.

4) The following restrictions apply to program prototypes, function prototypes, and method prototypes:

   a) The identification division shall not contain an ARITHMETIC clause.

b) The environment division shall not contain an object-computer paragraph.

c) The only clauses that may be specified in the SPECIAL-NAMES paragraph are the LOCALE clause, the CURRENCY clause, and the DECIMAL-POINT clause.

d) The environment division shall not contain an input-output section.

e) The data division may contain only a linkage section.

f) The procedure division shall contain only a procedure division header.

5) Compiler directives may appear in a structured compilation group as specified in 7.3, Compiler directives.

10.6.3 General rules

1) Compilation of a program prototype definition or a function prototype definition generates information required for the external repository, as specified in 8.13, External repository.

10.7 End markers

End markers indicate the end of a definition.

10.7.1 General format

```
END { PROGRAM program-prototype-name-1
    | PROGRAM program-name-1
    | CLASS class-name-1
    | FACTORY
    | FUNCTION function-prototype-name-1
    | FUNCTION user-function-name-1
    | OBJECT
    | METHOD [ method-name-1 ]
    | INTERFACE interface-name-1 } .
```

10.7.2 Syntax rules

1) An end marker shall be present in every source unit that contains, is contained in, or precedes another source unit.

2) Program-name-1 shall be identical to the program-name declared in a preceding PROGRAM-ID paragraph.

3) If a PROGRAM-ID paragraph declaring a specific program-name is stated between the PROGRAM-ID paragraph and the END PROGRAM marker for program-name-1, then an END PROGRAM marker referencing program-name shall precede the END PROGRAM marker referencing program-name-1.

4) Class-name-1 shall be identical to the class-name declared in the corresponding CLASS-ID paragraph.

5) Method-name-1 shall be identical to the method-name declared in the corresponding METHOD-ID paragraph. If the PROPERTY phrase is specified in the METHOD-ID paragraph, method-name-1 shall be omitted.

6) Interface-name-1 shall be identical to the interface-name declared in the corresponding INTERFACE-ID paragraph.

7) User-function-name-1 shall be identical to the user-function-name declared in the corresponding FUNCTION-ID paragraph.

8) Program-prototype-name-1 shall be identical to the program-prototype-name declared in the corresponding PROGRAM-ID paragraph.

9) Function-prototype-name-1 shall be identical to the function-prototype-name declared in the corresponding FUNCTION-ID paragraph.

10.7.3 General rules

1) An end marker indicates the end of the specified source unit.

11 Identification division

11.1 General

The identification division identifies the program, function, class, factory object, object, method, or interface.

The paragraph header identifies the type of information contained in the paragraph.

11.2 Identification division structure

11.2.1 General format

```
[ IDENTIFICATION DIVISION. ]
{ program-id-paragraph    }
{ function-id-paragraph   }
{ class-id-paragraph      }
{ factory-paragraph       }
{ object-paragraph        }
{ method-id-paragraph     }
{ interface-id-paragraph  }
[ options-paragraph ]
```

where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
class-id-paragraph	11.3, CLASS-ID paragraph
factory-paragraph	11.4, FACTORY paragraph
function-id-paragraph	11.5, FUNCTION-ID paragraph
interface-id-paragraph	11.6, INTERFACE-ID paragraph
method-id-paragraph	11.7, METHOD-ID paragraph
object-paragraph	11.8, OBJECT paragraph
options-paragraph	11.9, OPTIONS paragraph
program-id-paragraph	11.10, PROGRAM-ID paragraph

11.3 CLASS-ID paragraph

The CLASS-ID paragraph indicates that this identification division is introducing a class definition and specifies the name that identifies the class and assigns class attributes to the class.

11.3.1 General format

```
CLASS-ID. class-name-1 [ AS literal-1 ] [ IS FINAL ]
    [ INHERITS FROM { class-name-2 } ... ]
    [ USING { parameter-name-1 } ... ] .
```

11.3.2 Syntax rules

1) Literal-1 shall be an alphanumeric literal or a national literal and shall not be a figurative constant.

2) Class-name-2 shall be the name of a class specified in the REPOSITORY paragraph of this source element.

3) Class-name-2 shall not be the name of the class declared by this class definition.

4) Class-name-2 shall not inherit from class-name-1 directly or indirectly.

5) Class-name-2 shall not be the name of a class defined with the FINAL clause.

6) If two or more different methods with the same name are inherited, none of them may be specified with the FINAL clause. If the same method is inherited from one superclass through two or more intermediate superclasses, it may be specified with the FINAL clause.

7) A given class name shall not appear more than once in an INHERITS clause.

8) Parameter-name-1 shall be a name specified in a class-specifier or an interface-specifier in the REPOSITORY paragraph of this class definition.

9) A given parameter-name shall not appear more than once in a USING clause.

11.3.3 General rules

1) Class-name-1 names the class declared by this class definition. However, literal-1, if specified, is the name of the class that is externalized to the operating environment.

2) The INHERITS clause specifies the names of classes that are inherited by class-name-1 according to 9.3.9, Class inheritance.

3) If the FINAL clause is specified, the class shall not be the superclass for any other class.

4) If the same class is inherited more than once, then only one copy of the data for that class is added to class-name-1.

> NOTE While the same class cannot be directly inherited more than once, a class may be indirectly inherited multiple times. For example, suppose class D inherits from classes B and C, and classes B and C both inherit from class A. In this example, class D indirectly inherits class A twice, once as a superclass of B, and once as a superclass of C.

5) The USING clause specifies that this is a parameterized class. Parameter-name-1 is the name given to the formal parameter. See 9.3.12, Parameterized classes, for details of the behavior of a parameterized class.

6) Parameter-name-1 may be specified within this class definition only where a class-name or an interface-name is permitted.

11.4 FACTORY paragraph

The FACTORY paragraph indicates that this identification division is introducing a factory definition.

11.4.1 General format

FACTORY. [IMPLEMENTS { interface-name-1 } ... .]

11.4.2 Syntax rules

1) Interface-name-1 shall be the name of an interface specified in the REPOSITORY paragraph of the containing class definition.

2) Each method prototype in each implemented interface shall be such that the factory interface of this class conforms to all implemented interfaces.

11.4.3 General rules

1) The IMPLEMENTS clause specifies the names of the interfaces that are implemented by the factory object of the containing class according to 9.3.11, Interface implementation.

2) A factory object implements an interface int-1 in the following cases:

   a) the factory object is defined with an IMPLEMENTS clause specifying int-1,

   b) the factory object implements an interface that inherits int-1,

   c) the class containing the factory object inherits a class whose factory object implements int-1.

11.5 FUNCTION-ID paragraph

The FUNCTION-ID paragraph specifies the name by which a function is identified and assigns selected attributes to that function.

11.5.1 General format

Format 1 (definition):

FUNCTION-ID. user-function-name-1 [AS literal-1] .

Format 2 (prototype):

FUNCTION-ID. function-prototype-name-1 [AS literal-1] IS PROTOTYPE.

11.5.2 Syntax rules

1) Literal-1 shall be an alphanumeric literal or a national literal and shall be neither a figurative constant nor a zero-length literal.

11.5.3 General rules

FORMAT 1

1) User-function-name-1 names the function declared by this function definition. However, literal-1, if specified, is the name of the function that is externalized to the operating environment.

FORMAT 2

2) Function-prototype-name-1 names the function prototype declared by this definition. However, literal-1, if specified, is the name of the function prototype that is externalized to the operating environment.

11.6 INTERFACE-ID paragraph

The INTERFACE-ID paragraph indicates that this identification division is introducing an interface definition, specifies the name that identifies the interface, and assigns interface attributes to the interface.

11.6.1 General format

```
INTERFACE-ID.  interface-name-1  [ AS literal-1 ]
    [ INHERITS FROM { interface-name-2 }  ... ]
    [ USING { parameter-name-1 }  ... ] .
```

11.6.2 Syntax rules

1) Literal-1 shall be an alphanumeric literal or a national literal and shall be neither a figurative constant nor a zero-length literal.

2) Interface-name-2 shall be the name of an interface specified in the REPOSITORY paragraph of this source element.

3) Interface-name-2 shall not inherit directly or indirectly from interface-name-1.

4) Parameter-name-1 shall be a name specified in a class-specifier or an interface-specifier in the REPOSITORY paragraph of this interface definition.

5) If a given method-name is inherited from more than one interface, the method prototype in each inherited interface shall be such that this interface conforms to all inherited interfaces.

6) A given interface-name shall not appear more than once in an INHERITS clause.

7) A given parameter-name shall not appear more than once in a USING clause.

11.6.3 General rules

1) Interface-name-1 names the interface declared by this interface definition. However, literal-1, if specified, is the name of the interface that is externalized to the operating environment.

2) The INHERITS clause specifies the names of interfaces that are inherited by interface-name-1 according to 9.3.10, Interface inheritance.

3) The USING clause specifies that this is a parameterized interface. Parameter-name-1 is the name given to the formal parameter.

4) Parameter-name-1 shall be specified within this interface definition only where a class-name or an interface-name is permitted.

11.7 METHOD-ID paragraph

The METHOD-ID paragraph indicates that this identification division is introducing a method definition, specifies the name that identifies the method or method prototype, and assigns method attributes to the method.

11.7.1 General format

```
METHOD-ID. { method-name-1 [ AS literal-1 ]                          }
           { { GET } PROPERTY property-name-1                        } [ OVERRIDE ] [ IS FINAL ] .
           { { SET }                                                 }
```

11.7.2 Syntax rules

1) Literal-1 shall be an alphanumeric literal or a national literal and shall be neither a figurative constant nor a zero-length literal.

2) The OVERRIDE phrase shall not be specified in a method prototype.

3) If the OVERRIDE phrase is specified, there shall be a method with the same method resolution signature as the method declared by this method definition defined in a superclass. The method in the superclass shall not be defined with the FINAL clause.

4) If the OVERRIDE phrase is not specified:

   a) if this method definition is contained in a class definition, no inherited method shall have the same method resolution signature as the method declared by this method definition

   b) if this method definition is contained in an interface definition, no inherited method prototype shall have the same method resolution signature as the method prototype declared by this method definition.

5) If property-name-1 is specified as a data-name in the working-storage section of the containing object definition, the PROPERTY clause shall not be specified in the data description entry of that data-name.

6) If the GET phrase is specified, then the method shall have no USING phrase parameters specified in the procedure division header and shall have a single RETURNING phrase. The returning item shall not be an object reference described with the ACTIVE-CLASS phrase.

7) If the SET phrase is specified, then the method shall have a single USING parameter specified in the procedure division header and no RETURNING phrase. The USING parameter shall not be an object reference described with the ACTIVE-CLASS phrase.”

8) The FINAL clause shall not be specified in a method prototype.

9) If method-name-1 or literal-1 is the same as a method-name inherited or implemented by the containing definition, the parameter declarations, returning item, and exceptions that may be raised on the procedure division header shall obey the rules of conformance according to 9.3.8.1.2, Conformance between interfaces, such that the interface described by the factory or instance definition containing this method definition conforms to the interface described by the factory or instance definition containing the inherited or implemented method definition.

11.7.3 General rules

1) The name of the method declared by this method definition is determined as follows:

a) If the PROPERTY clause is specified, the name is implementor-defined.

b) Otherwise, the name is method-name-1. However, literal-1, if specified, is the name of method that is externalized to the operating environment.

2) The OVERRIDE phrase indicates that this method overrides the inherited method.

3) The FINAL clause indicates that this method shall not be overridden in any subclasses.

4) The name of this method may be referenced in the invocation of a method for an object of the class in which this method is defined.

5) If a given user-defined word is defined in the data division of this method definition and in the data division of the containing object definition, the use of that word in this method refers to the declaration in this method. The declaration in the containing object definition is inaccessible to this method.

6) If the GET phrase is specified, this method is a get property method for property-name-1.

7) If the SET phrase is specified, this method is a set property method for property-name-1.

11.8 OBJECT paragraph

The OBJECT paragraph indicates that this identification division is introducing an instance object definition.

11.8.1 General format

OBJECT. [IMPLEMENTS { interface-name-1 } ... .]

11.8.2 Syntax rules

1) Interface-name-1 shall be the name of an interface specified in the REPOSITORY paragraph of the containing class definition.

2) Each method prototype in each implemented interface shall be such that the object interface of this class conforms to all implemented interfaces.

11.8.3 General rules

1) The IMPLEMENTS clause specifies the names of the interfaces that are implemented by the object of the containing class according to 9.3.11, Interface implementation.

2) An instance object implements an interface intf-1 in the following cases:

   a) the instance object is defined with an IMPLEMENTS clause specifying intf-1,

   b) the instance object implements an interface that inherits intf-1,

   c) the class containing the instance object inherits a class whose instance object implements intf-1.

11.9 OPTIONS paragraph

The OPTIONS paragraph specifies information for use by the compiler in generating executable code for a source unit.

11.9.1 General format

```
OPTIONS.
    [ arithmetic-clause ]
    [ default-rounded-clause ]
    [ entry-convention-clause ]
    [ intermediate-rounding-clause ] .
```

11.9.2 Syntax rules

1) One of the separator periods in the general format may be omitted if none of the clauses in the OPTIONS paragraph is specified.

11.9.3 General Rules

1) The clauses in the OPTIONS paragraph apply to the source element in which they are specified and to all source elements contained in that source element unless overridden by a clause in an OPTIONS paragraph in a contained source element.

11.9.4 ARITHMETIC clause

The ARITHMETIC clause specifies the method used in developing the intermediate results.

11.9.4.1 General format

```
ARITHMETIC IS { NATIVE           }
              { STANDARD         }
              { STANDARD-BINARY  }
              { STANDARD-DECIMAL }
```

11.9.4.2 General rules

1) If the NATIVE phrase is specified, the techniques used in handling arithmetic expressions and intrinsic functions shall be those specified by the implementor, and the techniques used in handling arithmetic statements and the SUM clause shall be those specified for native arithmetic in 8.8.1.2, Native arithmetic.

2) If the STANDARD phrase is specified, the techniques used in handling arithmetic expressions, arithmetic statements, the SUM clause, and certain integer and numeric functions shall be those specified for standard arithmetic in 8.8.1.3, Standard arithmetic.

   NOTE 1 Standard arithmetic is an obsolete feature of standard COBOL.

   NOTE 2 Implementors are strongly encouraged to provide support for the STANDARD-DECIMAL phrase of the ARITHMETIC clause.

3) If the STANDARD-BINARY phrase is specified, the techniques used in handling arithmetic expressions, arithmetic statements, the SUM clause, and integer and numeric functions shall be as described for standard-binary arithmetic in 8.8.1.4, Standard-binary arithmetic.

4) If the STANDARD-DECIMAL phrase is specified, the techniques used in handling arithmetic expressions, arithmetic statements, the SUM clause, and integer and numeric functions shall be as described for standard-decimal arithmetic in 8.8.1.5, Standard-decimal arithmetic.

5) If the ARITHMETIC clause is not specified in this source element or a containing source element, it is as if the ARITHMETIC clause were specified with the NATIVE phrase.

11.9.5 DEFAULT ROUNDED clause

The DEFAULT ROUNDED clause specifies the type of rounding that applies when ROUNDED phrases of arithmetic statements, SOURCE clauses, and SUM clauses do not specify a rounding mode.

11.9.5.1 General format

```
DEFAULT ROUNDED MODE IS { AWAY-FROM-ZERO
                        | NEAREST-AWAY-FROM-ZERO
                        | NEAREST-EVEN
                        | NEAREST-TOWARD-ZERO
                        | PROHIBITED
                        | TOWARD-GREATER
                        | TOWARD-LESSER
                        | TRUNCATION }
```

11.9.5.2 General rules

1) The DEFAULT ROUNDED clause specifies the type of rounding that applies to any explicit ROUNDED phrase that does not include an explicit MODE phrase. The meaning of the rounding types associated with the DEFAULT ROUNDED clause is described in 14.7.3, ROUNDED phrase.

2) If the DEFAULT ROUNDED clause is not specified, DEFAULT ROUNDED MODE IS NEAREST-AWAY-FROM-ZERO is implied.

NOTE NEAREST-AWAY-FROM-ZERO is the rounding mode provided by the ROUNDED phrase in previous COBOL standards.

11.9.6 ENTRY-CONVENTION clause

The ENTRY-CONVENTION clause specifies the information to be used for activating with a runtime element.

11.9.6.1 General format

```
ENTRY-CONVENTION IS { COBOL
                    | entry-convention-name-1 }
```

11.9.6.2 Syntax rules

1) The ENTRY-CONVENTION clause may be specified only in a class definition, a function definition, a function-prototype definition, an interface definition, a program prototype definition, or a program definition that is not contained within another program.

11.9.6.3 General rules

1) The ENTRY-CONVENTION clause specifies the convention to be used for activating the runtime element corresponding to the source element in which this clause is specified.

   NOTE All information required to interact successfully with a runtime element should be made available to the compiler via this entry convention association; this includes items such as name case sensitivity, how arguments are passed, and stack management.

2) When COBOL is specified, the naming convention and mapping of method-names and program-names are as specified in 8.3.1.1.1, User-defined words; other aspects of the entry convention are implementor-defined.

3) When entry-convention-name-1 is specified, the meaning of the entry convention is implementor-defined.

4) When ENTRY-CONVENTION is not specified, the entry convention used is as follows:

   a) If a class definition includes the INHERITS clause, the entry convention is inherited from the first class specified in the INHERITS clause.

   b) If an interface definition includes the INHERITS clause, the entry convention is inherited from the first interface specified in the INHERITS clause.

   c) In all other cases, the entry convention is COBOL.

11.9.7 INTERMEDIATE ROUNDING clause

The INTERMEDIATE ROUNDING clause specifies the rounding rules that are to be applied during the production of intermediate results in arithmetic statements and arithmetic expressions.

11.9.7.1 General format

```
INTERMEDIATE ROUNDING IS { NEAREST-AWAY-FROM-ZERO }
                         { NEAREST-EVEN           }
                         { PROHIBITED             }
                         { TRUNCATION             }
```

11.9.7.2 General rules

1) If native arithmetic is in effect, the rounding rules that apply to intermediate data items are defined by the implementor.

2) If standard arithmetic is in effect, the rounding rules that apply to standard intermediate data items are specified in 8.8.1.3.1.3, Rounding rules for standard arithmetic.

3) If standard-binary arithmetic is in effect:

   a) If the INTERMEDIATE ROUNDING clause is not specified, the NEAREST-EVEN clause is implied.

   b) The NEAREST-AWAY-FROM-ZERO phrase shall not be specified.

   c) If the NEAREST-EVEN phrase is specified or implied, the intermediate data item shall be rounded to the nearest value that can be represented in a binary128 format in which the rightmost bit is zero.

   d) If the PROHIBITED phrase is specified and the intermediate value cannot be represented exactly in a binary128 format, the EC-SIZE-TRUNCATION exception condition is set to exist and the results of the operation are undefined.

e) If the TRUNCATION phrase is specified and an intermediate value cannot be exactly represented in a binary128 format, the value shall be rounded to the nearest value that can be represented in that format in which the magnitude is smaller than the intermediate value and in which the rightmost bit of the significand is zero.

4) If standard-decimal arithmetic is in effect:

a) If the INTERMEDIATE ROUNDING clause is not specified, the NEAREST-AWAY-FROM-ZERO clause is implied.

b) If the NEAREST-AWAY-FROM-ZERO phrase is specified, and an intermediate data item cannot be exactly represented in a decimal128 format, the value is rounded to the nearest value that can be represented in that format. If two such values are equally near, the value whose magnitude is larger is used.

c) If the NEAREST-EVEN phrase is specified or implied, the intermediate data item shall be rounded to the nearest value that can be represented in a decimal128 format. If two such values are equally near, the value in which the rightmost digit of the significand is even shall be delivered.

d) If the PROHIBITED phrase is specified and an intermediate value cannot be exactly represented in a decimal128 format, the EC-SIZE-TRUNCATION exception condition is set to exist and the results of the operation are undefined.

e) If the TRUNCATION phase is specified and an intermediate data item cannot be exactly represented in a decimal128 format, the value shall be the nearest value in that format whose magnitude is not greater than the intermediate value.

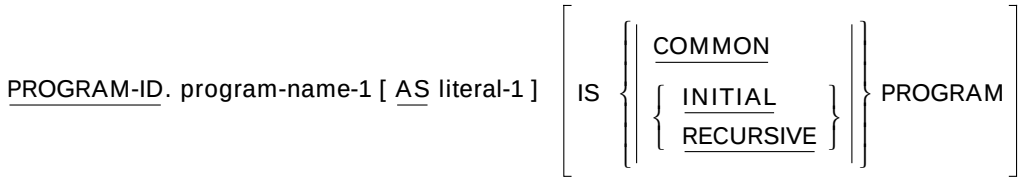
11.10 PROGRAM-ID paragraph

The PROGRAM-ID paragraph specifies the name by which a program is identified and assigns selected program attributes to that program.

The PROGRAM-ID paragraph specifies the name by which a program prototype is identified.

11.10.1 General format

Format 1 (definition):



Format 2 (prototype):

PROGRAM-ID. program-prototype-name-1 [AS literal-1] IS PROTOTYPE .

11.10.2 Syntax rules

ALL FORMATS

1) Literal-1 shall be an alphanumeric literal or a national literal and shall be neither a figurative constant nor a zero-length literalt.

FORMAT 1

2) Literal-1 shall not be specified in a program that is contained within another program.

3) A program contained within another program shall not be assigned the same name as that of any other program contained within the outermost program that contains this program.

4) The COMMON clause may be specified only if the program is contained within another program.

5) The INITIAL clause shall not be specified if any program that directly or indirectly contains this program is a recursive program.

6) The RECURSIVE clause shall not be specified if any program that directly or indirectly contains this program is an initial program.

11.10.3 General rules

FORMAT 1

1) Program-name-1 names the program declared by this program definition. Literal-1, if specified, is the name of the program that is externalized to the operating environment.

2) The COMMON clause specifies that the program is common. A common program is contained within another program but may be called from programs other than that containing it. (See 8.4.5, Scope of names.)

3) The INITIAL clause specifies that the program is initial. When an initial program is activated, the data items and file connectors contained in it and any program contained within it are set to their initial states.

4) The RECURSIVE clause specifies that the program and any programs contained within it are recursive. The program may be called while it is active and may call itself. If the RECURSIVE clause is not specified in a program or implied for a program, the program shall not be called while it is active.

5) Additional rules concerning initial and recursive programs are given in 8.6.5, Common, initial, and recursive attributes.

FORMAT 2

6) Program-prototype-name-1 identifies the program prototype. However, literal-1, if specified, is the name of the program prototype that is externalized to the operating environment.

12 Environment division

12.1 General

The environment division specifies those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer. This division allows specification of the configuration of the compiling computer and the object computer. In addition, information relative to input-output control, special hardware characteristics, and control techniques can be given.

12.2 Environment division structure

12.2.1 General format

ENVIRONMENT DIVISION.
[configuration-section]
[input-output-section]

12.3 Configuration section

The configuration section specifies aspects of the data processing system that are dependent on the specific system as well as special control techniques and a means of associating a local name with an external resource. This section is divided into paragraphs:

— the SOURCE-COMPUTER paragraph, which describes the computer configuration on which the source element is compiled;

— the OBJECT-COMPUTER paragraph, which describes the computer configuration on which the runtime module produced by the compiler is to be run;

— the SPECIAL-NAMES paragraph, which provides a means for specifying the currency sign, choosing the decimal point, specifying symbolic-characters, relating system-names to user-specified mnemonic-names, relating alphabet-names to character sets or collating sequences, and relating class-names to sets of characters; and

— the REPOSITORY paragraph, which provides a means for associating a local name with an external resource and specifying which intrinsic function names become reserved words for this source unit.

12.3.1 General format

```
CONFIGURATION SECTION.
[ source-computer-paragraph ]
[ object-computer-paragraph ]
[ special-names-paragraph ]
[ repository-paragraph ]
```

12.3.2 Syntax rules

1) The configuration section shall not be specified in a program that is contained within another program.

2) The configuration section shall not be specified in a method definition.

3) The SOURCE-COMPUTER, OBJECT-COMPUTER, and REPOSITORY paragraphs shall not be specified in a factory definition or an instance definition.

12.3.3 General rules

1) The entries explicitly or implicitly specified in the configuration section of a source unit that contains other source units apply to each directly or indirectly contained source unit.

12.3.4 SOURCE-COMPUTER paragraph

The SOURCE-COMPUTER paragraph provides a means of describing the computer upon which the compilation unit is to be compiled.

12.3.4.1 General format

SOURCE-COMPUTER. [computer-name-1] .

12.3.4.2 Syntax rules

1) If computer-name-1 is not specified, the second period in the general format may be omitted.

12.3.4.3 General rules

1) All clauses of the SOURCE-COMPUTER paragraph apply to the source unit in which they are explicitly or implicitly specified and to any source unit contained within that source unit.

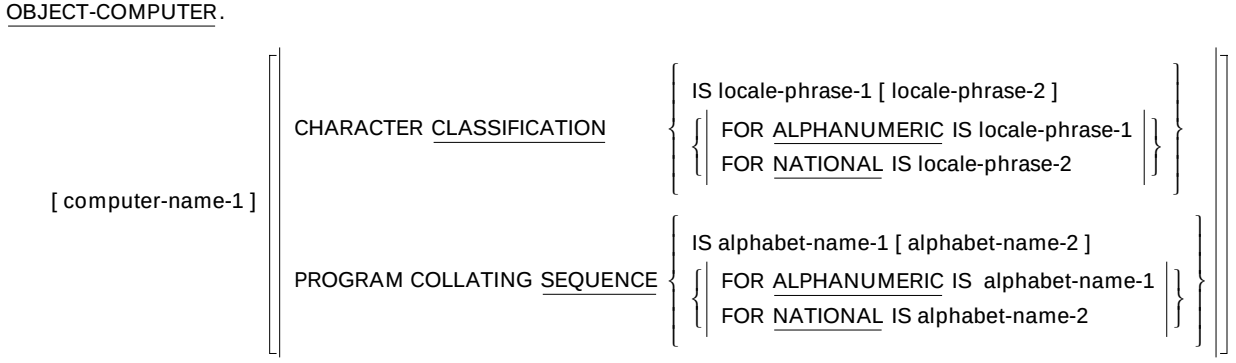
2) When the SOURCE-COMPUTER paragraph is not specified and the source unit is not contained within a source unit including a SOURCE-COMPUTER paragraph, the computer on which the source unit is being compiled is the source computer.

3) When the SOURCE-COMPUTER paragraph is specified, but computer-name-1 is not specified, the computer upon which the source unit is being compiled is the source computer.

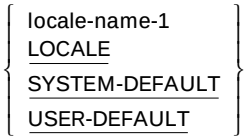
12.3.5 OBJECT-COMPUTER paragraph

The OBJECT-COMPUTER paragraph provides a means of describing the computer on which the runtime module created by the compiler is to be executed.

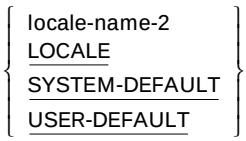
12.3.5.1 General format



where locale-phrase-1 is:



where locale-phrase-2 is:



12.3.5.2 Syntax rules

1) Alphabet-name-1 shall reference an alphabet that defines an alphanumeric collating sequence.

2) Alphabet-name-2 shall reference an alphabet that defines a national collating sequence.

3) Locale-name-1 and locale-name-2 shall be locale names defined in the SPECIAL-NAMES paragraph.

4) If neither computer-name-1 nor any of the optional clauses is specified, the second period in the general format may be omitted.

12.3.5.3 General rules

1) All clauses of the OBJECT-COMPUTER paragraph apply to the source unit in which they are explicitly or implicitly specified and to any source unit contained within that source unit.

2) Computer-name-1 may provide a means for identifying equipment configuration, in which case computer-name-1 and its implied configuration are specified by each implementor.

3) When the OBJECT-COMPUTER paragraph is specified, but computer-name-1 is not specified, the object computer is defined by the implementor.

4) When the OBJECT-COMPUTER paragraph is not specified and the source unit is not contained within a source unit including an OBJECT-COMPUTER paragraph, the object computer is defined by the implementor.

5) When the CHARACTER CLASSIFICATION clause is specified, the initial character classification is as follows:

   a) If locale-name-1 is specified, the initial alphanumeric character classification is the character classification associated with locale-name-1.

   b) If LOCALE is specified as locale-phrase-1, the initial alphanumeric character classification is the character classification associated with the current locale.

   c) If SYSTEM-DEFAULT is specified as locale-phrase-1, the initial alphanumeric character classification is the character classification associated with the system default locale.

   d) If USER-DEFAULT is specified as locale-phrase-1, the initial alphanumeric character classification is the character classification associated with the user default locale.

   e) If locale-phrase-1 is not specified, the initial alphanumeric character classification is the character classification associated with the computer's coded character set in effect for alphanumeric characters at runtime.

   f) If locale-name-2 is specified, the initial national character classification is the character classification associated with locale-name-2.

   g) If LOCALE is specified as locale-phrase-2, the initial national character classification is the character classification associated with the current locale.

   h) If SYSTEM-DEFAULT is specified as locale-phrase-2, the initial national character classification is the character classification associated with the system default locale.

   i) If USER-DEFAULT is specified as locale-phrase-2, the initial national character classification is the character classification associated with the user default locale.

   j) If locale-phrase-2 is not specified, the initial national character classification is the character classification associated with the computer's coded character set in effect for national characters at runtime.

6) When the CHARACTER CLASSIFICATION clause is not specified and the source unit is not contained within a source unit for which a CHARACTER CLASSIFICATION clause is specified, the initial character classifications are the character classifications associated with the computer's coded character set in effect for alphanumeric and national characters at runtime.

7) When the CHARACTER CLASSIFICATION clause is specified, the cultural convention specification LC_CTYPE from the specified locales are used for:

   a) the uppercase and lowercase mappings of characters for the UPPER-CASE and LOWER-CASE intrinsic functions.

   b) the classification of characters for class tests ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, and for the class test specifying an alphabet-name that is associated with a locale in the SPECIAL-NAMES paragraph.

8) The character classifications explicitly or implicitly established by the OBJECT-COMPUTER paragraph are effective with the initial state of the runtime modules to which they apply. If locale-name-1 or locale-name-2 is specified, the associated character classification is defined by category LC_CTYPE in the locale identified by that locale-name.

9) When the PROGRAM COLLATING SEQUENCE clause is specified, the initial alphanumeric program collating sequence is the collating sequence associated with alphabet-name-1 and the initial national program collating sequence is the collating sequence associated with alphabet-name-2. When alphabet-name-1 is not specified, the initial alphanumeric program collating sequence is the native alphanumeric collating sequence; when alphabet-name-2 is not specified, the initial national program collating sequence is the native national collating sequence.

10) When the PROGRAM COLLATING SEQUENCE clause is not specified and the source unit is not contained within a source unit for which a PROGRAM COLLATING SEQUENCE clause is specified, the initial program collating sequences are the native alphanumeric collating sequence and the native national collating sequence.

11) The alphanumeric program collating sequence and national program collating sequence are used to determine the truth value of any alphanumeric comparisons and national comparisons, respectively, that are:

    a) Explicitly specified in relation conditions.

    b) Explicitly specified in condition-name conditions.

    c) Implicitly specified by the presence of a CONTROL clause in a report description entry.

    When alphabet-name-1 or alphabet-name-2, or both, is associated with a locale, locale category LC_COLLATE is used to carry out these comparisons.

12) The alphanumeric program collating sequence and national program collating sequence explicitly or implicitly established by the OBJECT-COMPUTER paragraph are effective with the initial state of the runtime modules to which they apply. If alphabet-name-1 or alphabet-name-2 references a locale, the associated collating sequence is defined by category LC_COLLATE in the specific locale associated with that alphabet-name or, if none, in the locale current at the time the collating sequence is used at runtime

13) The alphanumeric program collating sequence and national program collating sequence are applied to alphanumeric and national sort or merge keys, respectively, unless the sort or merge collating sequence has been modified by execution of a SET statement or a COLLATING SEQUENCE phrase is specified in the respective SORT or MERGE statement.

12.3.6 SPECIAL-NAMES paragraph

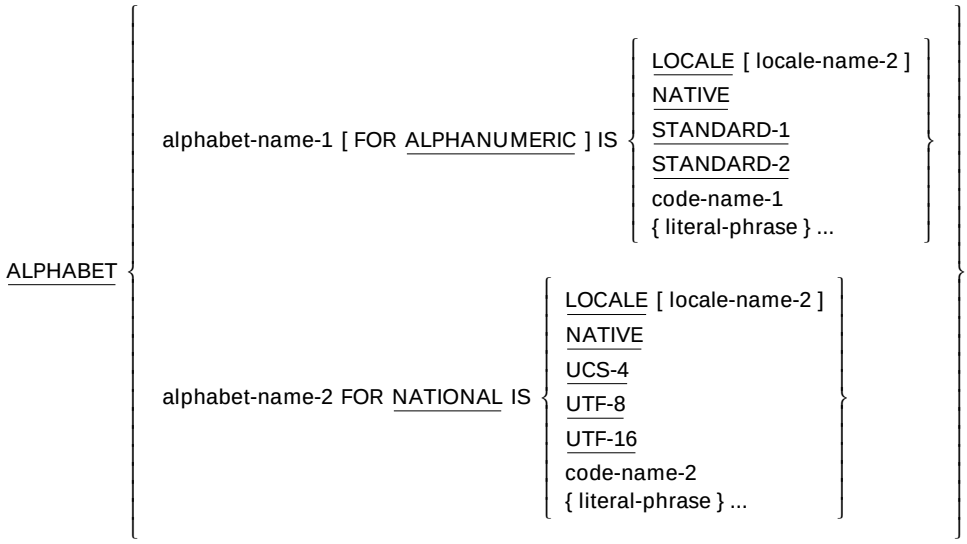
The SPECIAL-NAMES paragraph provides a means for:

- specifying currency signs and symbols,
- choosing the decimal point,
- specifying symbolic-characters,
- relating any-length-structure-names to the structure of an any-length elementary item,
- relating system-names to user-specified mnemonic-names,
- relating locale-names to the external identification of locales,
- relating alphabet-names to character sets or collating sequences or both,
- relating class-names to a set of characters,
- relating an ordering table name to a standard ordering table,
- relating a data item to the cursor position of a character addressable terminal, and
- relating a data item to the status of a terminal input-output operation.

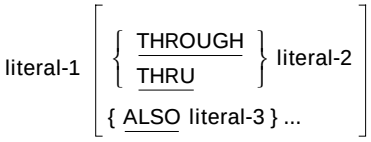
12.3.6.1 General format

```
SPECIAL-NAMES.
[ alphabet-name-clause ] ...
[ any-length-structure-clause ] ...
[ CLASS class-name-1 [ FOR { ALPHANUMERIC } ]                                  ]
[                          { NATIONAL      }                                   ]
[                                                                              ] ...
[      IS { literal-5 [ { THROUGH } literal-6 ] } ... [ IN alphabet-name-4 ]   ]
[         { literal-5 [ { THRU    }           ] }                              ]
[ CRT STATUS IS data-name-2 ]
[ CURRENCY SIGN IS literal-7 [ WITH PICTURE SYMBOL literal-8 ] ] ...
[ CURSOR IS data-name-1 ]]
[ DECIMAL-POINT IS COMMA ]
[ LOCALE locale-name-1 IS { external-locale-name-1 } ] ...
[                         { literal-4              } ]
[ switch-name-1                                                                ]
[      { IS mnemonic-name-1 [ { ON STATUS IS condition-name-1  } ]             }
[      {                    [ { OFF STATUS IS condition-name-2 } ]             }
[      { [ { ON STATUS IS condition-name-1  } ]                                 }
[      { [ { OFF STATUS IS condition-name-2 } ]                                 }
[ feature-name-1 IS mnemonic-name-2                                            ]
[ device-name-1 IS mnemonic-name-3                                             ]
[ symbolic-characters-clause ] ...
[ ORDER TABLE ordering-name-1 IS literal-9 ] .
```

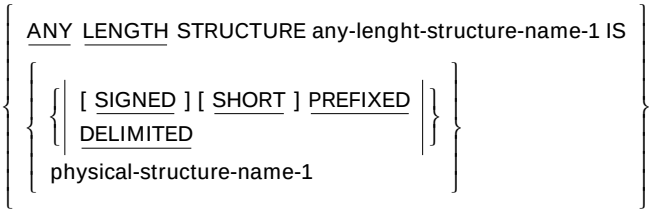
where alphabet-name-clause is:



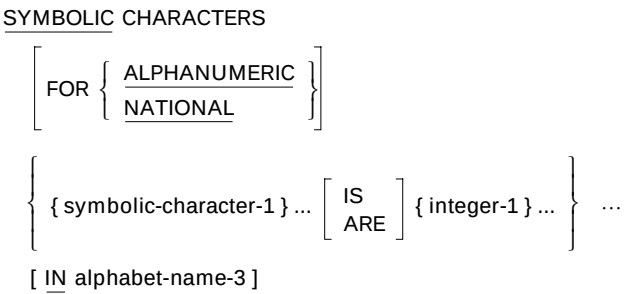
where literal-phrase is:



were any-length-structure-clause is:



where symbolic-characters-clause is:



12.3.6.2 Syntax rules

1) At the outer level of a source unit that defines a class, the CURSOR and CRT STATUS clauses shall not be specified.

2) In a factory definition or instance definition, the only clauses that may be specified are the CURSOR and CRT STATUS clauses.

3) In an interface definition, the ALPHABET clause, the CURRENCY clause, the DECIMAL-POINT clause, and the LOCALE clause are the only permitted clauses.

4) In a program definition, data-name-1 and data-name-2 shall be described with the GLOBAL clause if the program definition contains one or more program definitions.

5) Mnemonic-name-1 may be specified only in a SET statement.

6) Mnemonic-name-2 may be specified only in the WRITE statement. The implementor may specify additional restrictions on the use of mnemonic-names that reference specific feature-names.

7) Mnemonic-name-3 may be specified only in the ACCEPT and DISPLAY statements. The implementor may specify additional restrictions on the use of mnemonic-names that reference specific device-names.

8) The implementor shall specify the names that are available for switch-name-1, feature-name-1, and device-name-1.

9) Ordering-name-1 may be specified only in the STANDARD-COMPARE intrinsic function.

10) Literal-4 and literal-9 shall be alphanumeric or national literals.

11) Literal-1, literal-2, literal-3, literal-4, literal-5, literal-6, and literal-9 shall specify neither a symbolic-character figurative constant nor a zero-length literal.

12) The words THROUGH and THRU are equivalent.

13) When the ALPHABET clause is specified with neither the ALPHANUMERIC phrase nor the NATIONAL phrase, the ALPHANUMERIC phrase is implied.

14) When the ALPHABET clause is specified with a literal-phrase:

    a) A given character shall not be specified more than once in that ALPHABET clause.

    b) When the ALPHANUMERIC phrase is specified or implied:

        1. Each numeric literal shall be an unsigned integer and shall have a value within the range of one through the maximum number of characters in the native alphanumeric character set.

        2. Each noninteger literal shall be an alphanumeric literal.

        3. Each alphanumeric literal, when a THROUGH or ALSO phrase is specified, shall be one character in length.

        4. The number of characters specified shall not exceed the number of characters in the native alphanumeric character set.

    c) When the NATIONAL phrase is specified:

        1. Each numeric literal shall be an unsigned integer and shall have a value within the range of one through the maximum number of characters in the native national character set.

2. Each noninteger literal shall be a national literal.

3. Each national literal, when a THROUGH or ALSO phrase is specified, shall be one character in length.

4. The number of characters specified shall not exceed the number of characters in the native national character set.

15) The implementor shall specify the names supported for code-name-1 and code-name-2 in the ALPHABET clause, if any.

16) When the SYMBOLIC CHARACTERS clause is specified:

a) A given symbolic-character-1 may be specified only once within the SYMBOLIC CHARACTER clauses of this SPECIAL-NAMES paragraph.

b) The relationship between each symbolic-character-1 and the corresponding integer-1 is by position in the SYMBOLIC CHARACTERS clause. The first symbolic-character-1 is paired with the first integer-1; the second symbolic-character-1 is paired with the second integer-1; and so on.

c) There shall be a one-to-one correspondence between occurrences of symbolic-character-1 and occurrences of integer-1.

d) When neither the ALPHANUMERIC phrase nor the NATIONAL phrase is specified, the ALPHANUMERIC phrase is implied.

e) When the ALPHANUMERIC phrase is specified or implied:

1. When the IN phrase is specified, alphabet-name-3 shall reference an alphabet that defines an alphanumeric character set, and the ordinal position specified by integer-1 shall exist in that character set.

2. When the IN phrase is not specified, the ordinal position specified by integer-1 shall exist in the native alphanumeric character set.

f) When the NATIONAL phrase is specified:

1. When the IN phrase is specified, alphabet-name-3 shall reference an alphabet that defines a national character set, and the ordinal position specified by integer-1 shall exist in that character set.

2. When the IN phrase is not specified, the ordinal position specified by integer-1 shall exist in the native national character set.

g) Alphabet-name-3 shall not reference an alphabet specified with the LOCALE phrase.

17) When the CLASS clause is specified:

a) When neither the ALPHANUMERIC phrase nor the NATIONAL phrase is specified, the ALPHANUMERIC phrase is implied.

b) When the ALPHANUMERIC phrase is specified or implied:

1. When the IN phrase is specified, alphabet-name-4 shall reference an alphabet that defines an alphanumeric character set.

2. Literal-5, if numeric, shall be an unsigned integer and shall have a value within the range of one through the maximum number of characters in the native alphanumeric character set, or, when the IN phrase is specified, the maximum number of characters in the character set referenced by alphabet-name-4.

3. Each noninteger literal shall be an alphanumeric literal.

4. Each alphanumeric literal, when a THROUGH phrase is specified, shall be one character in length.

5. The number of characters specified shall not exceed the number of characters in the native alphanumeric character set or, when the IN phrase is specified, the number of characters in the character set referenced by alphabet-name-4.

c) When the NATIONAL phrase is specified:

1. When the IN phrase is specified, alphabet-name-4 shall reference an alphabet that defines a national character set.

2. Literal-5, if numeric, shall be an unsigned integer and shall have a value within the range of one through the number of characters in the native national character set, or, when the IN phrase is specified, the number of characters in the character set referenced by alphabet-name-4.

3. Each noninteger literal shall be a national literal.

4. Each national literal, when a THROUGH phrase is specified, shall be one character in length.

5. The number of characters specified shall not exceed the number of characters in the native national character set or, when the IN phrase is specified, the number of characters in the character set referenced by alphabet-name-4.

d) Alphabet-name-4 shall not reference an alphabet specified with the LOCALE phrase.

18) Literal-7 shall be an alphanumeric or national literal that is not a figurative constant.

19) If literal-7 is an alphanumeric or national literal in hexadecimal format, the PICTURE SYMBOL phrase shall be present in the CURRENCY SIGN clause.

NOTE Literals in hexadecimal format specified as literal-7 represent characters from the computer's runtime coded character set, and therefore the content of such literals is used at run time exactly as specified by the user. Because the content of such literals is interpreted at run time, the limitations as to which characters are permitted in a currency string at compile time do not apply to the alphanumeric or national characters represented by hexadecimal literals used as currency strings.

20) If the character specified as the currency symbol is not a character in the COBOL character repertoire, any equivalence between that character and any other character in the computer's compile-time coded character set is defined by the implementor. In all other cases, compile-time equivalence to that currency symbol is determined as specified in 8.1.2, COBOL character repertoire, general rules.

NOTE For example, N'M', N'm', 'M', and 'm' used as currency symbols are all equivalent.

21) No two CURRENCY SIGN clauses within a single source unit shall specify equivalent currency symbols unless they specify identical currency strings.

22) If the PICTURE SYMBOL phrase is not specified, literal-7 is both the currency string and the currency symbol. It shall consist of a single character from the computer's compile-time coded character set that is neither the same as nor equivalent to any of the following:

a) digits 0 through 9;

b) alphabetic characters A, B, C, D, E, N, P, R, S, V, X, Z, or their lowercase equivalents; or the space;

c) characters '+' '–' ',' '.' '*' '/' ';' '(' ')' '"' '='

23) If the PICTURE SYMBOL phrase is specified, literal-7 is the currency string and literal-8 is the associated currency symbol. Literal-7 may have any length and:

    a) shall contain at least one nonspace character and

    b) may consist of any characters from the computer's coded character set except for the digits 0 through 9 and the characters '+' '–' ',' '.' '*'.

24) Locale-name-2 shall be a locale-name defined by the LOCALE clause.

25) If a source unit does not contain a CURRENCY SIGN clause that specifies '$' as the currency symbol (either as literal-7 or literal-8), the clause CURRENCY SIGN '$' PICTURE SYMBOL '$' is implied for that source unit.

26) Literal-8 shall be an alphanumeric or national literal consisting of a single character from the computer's compile-time coded character set. It shall be neither a figurative constant nor a hexadecimal literal.

27) Literal-8 may be any character from the computer's coded character set except for the following:

    a) digits 0 through 9;

    b) alphabetic characters A, B, C, D, E, N, P, R, S, V, X, Z, or their lowercase equivalents; or the space;

    c) characters '+' '–' ',' '.' '*' '/' ';' '(' ')' '"' '='

28) If literal-7 is of class alphanumeric, the associated currency symbol may be used only to define a numeric-edited item with usage display. If literal-7 is of class national, the associated currency symbol may be used only to define a numeric-edited item with usage national.

29) Data-name-1 shall be described in the working-storage or local-storage section as either an elementary unsigned integer of 6 digits described implicitly or explicitly as usage display, or an alphanumeric group item consisting of two elementary unsigned integers of 3 digits described implicitly or explicitly as usage display.

30) Data-name-2 shall be described in the working-storage or local-storage section as an alphanumeric data item 4 characters in length.

31) One of the separator periods may be omitted if none of the clauses in the SPECIAL-NAMES paragraph is specified.

32) The implementor shall specify the names supported for physical-structure-name-1.

12.3.6.3 General rules

1) All clauses specified in the SPECIAL-NAMES paragraph of a source unit that contains other source units apply to each directly or indirectly contained source unit. The condition-names, mnemonic-names, locale-names, class-names, currency signs and symbols, alphabet-names, and symbolic-characters specified in the SPECIAL-NAMES paragraph of the containing source unit may be referenced from any directly or indirectly contained source unit.

2) Switch-name-1 identifies an implementor-defined external switch. The on status and the off status of an external switch may each be associated with a condition-name. The status of that switch can be interrogated by referencing the condition-names as specified in 8.8.4.1.5, Switch-status condition.

3) The status of an external switch may be altered by execution of a SET mnemonic-name statement that specifies as its operand the mnemonic-name associated with that switch. The implementor defines which external switches may be referenced by the SET statement.

4) The implementor defines the scope (program, run unit, etc.) of each external switch and any facility external to COBOL that may be used to modify the status of an external switch.

NOTE If the scope of an external switch is the run unit, each switch-name of such an external switch refers to one and only one such switch, the status of which is available to each runtime element functioning within that run unit.

5) When the LOCALE clause is specified, locale-name-1 references a locale identified by external-locale-name-1 or the value of literal-4. The implementor specifies the allowable external-locale-names and the allowable content of literal-4.

6) The implementor shall define the order of characters within the native alphanumeric coded character set and the native national coded character set, associating each character with an ordinal position within the character set.

7) The ALPHABET clause provides a means of relating a name to a specified coded character set or collating sequence, or both.

NOTE An alphabet-name referenced in the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph or in the COLLATING SEQUENCE phrase of a SORT or MERGE statement references a collating sequence. An alphabet-name referenced in the class condition, in the CLASS clause in the data division, in a SYMBOLIC CHARACTERS clause, or in the CODE-SET clause of a file description entry references a coded character set.

Table 7, Relationship of alphabet-name to coded character set and collating sequence, indicates for each operand of the ALPHABET clause whether the alphabet-name references a coded character set, a collating sequence, or both.

Table 7 — Relationship of alphabet-name to coded character set and collating sequence

ALPHABET clause operand	Coded character set	Collating sequence
LOCALE		Y
NATIVE	Y	Y
STANDARD-1	Y	Y
STANDARD-2	Y	Y
UCS-4	Y	Y
UTF-8	Y	
UTF-16	Y	
code-name-1	Y	Y
code-name-2	Y	Y
literal phrase	Y	Y

Legend:
Y in the Coded character set column indicates that the alphabet operand in the left column references a coded character set and a space means it does not.
Y in the Collating sequence column indicates that the alphabet operand in the left column references a collating sequence and a space means it does not.

When the ALPHABET clause is specified:

a) When the ALPHANUMERIC phrase is specified or implied, a coded character set referenced by alphabet-name-1 is an alphanumeric coded character set and a collating sequence referenced by alphabet-name-1 is an alphanumeric collating sequence.

b) When the NATIONAL phrase is specified, a coded character set referenced by alphabet-name-2 is a national coded character set and a collating sequence referenced by alphabet-name-2 is a national collating sequence.

c) When the STANDARD-1 phrase is specified, the referenced coded character set is defined in ISO/IEC 646 International Reference Version. When the STANDARD-2 phrase is specified, the referenced coded character set is defined in ISO/IEC 646; the implementor shall specify whether the International Reference Version or a national version is referenced, and shall specify the circumstances that define which national version is referenced. The collating sequence referenced by both the STANDARD-1 and STANDARD-2 phrases is defined by the order in which characters appear in ISO/IEC 646. Each character of the standard character set is associated with its corresponding character of the native alphanumeric character set. The implementor defines the correspondence between the characters of the standard character set and the characters of the native character set.

d) When the NATIVE phrase is specified:

   1. If the ALPHANUMERIC phrase is specified or implied, the native alphanumeric coded character set and native alphanumeric collating sequence are referenced;

   2. If the NATIONAL phrase is specified, the native national coded character set and native national collating sequence are referenced.

e) When the LOCALE phrase is specified, the collating sequence identified is defined by the locale referenced by locale-name-2 when specified, otherwise by the locale that is current at the time the collating sequence is used at runtime. When LOCALE is specified in the ALPHANUMERIC phrase, an alphanumeric collating sequence is identified; when specified in the NATIONAL phrase, a national collating sequence is identified.

f) When the UCS-4 phrase is specified, the coded character set referenced is that specified in ISO/IEC 10646 as UCS-4. Each character of UCS-4 is associated with a corresponding character of the native national character set. The implementor shall specify the correspondence between the characters of UCS-4 and the characters of the native national character set. The collating sequence referenced by the UCS-4 phrase is defined by the order in which characters appear in ISO/IEC 10646.

g) When the UTF-8 phrase is specified, the coded character set referenced is that specified in ISO/IEC 10646 as UTF-8. Each character of ISO/IEC 10646 UTF-8 is associated with a corresponding character of the native national character set. The association is the same as the association for the UCS-4 character from which the UTF-8 character was transformed.

h) When the UTF-16 phrase is specified, the coded character set referenced is that specified in ISO/IEC 10646 as UTF-16. Each character of ISO/IEC 10646 UTF-16 is associated with a corresponding character of the native national character set. The implementor shall specify the correspondence between the characters of ISO/IEC 10646 UTF-16 and the characters of the native national character set.

i) When code-name-1 is specified, the alphanumeric coded character set and collating sequence referenced are defined by the implementor. The implementor shall specify the ordinal number of each character for use when code-name-1 references a coded character set and the collating position of each character for use when code-name-1 references a collating sequence. The implementor also shall specify the correspondence between characters of the alphanumeric coded character set specified by code-name-1 and the characters of the native alphanumeric coded character set.

   The coded character set referenced by code-name-1 is statically defined.

j) When code-name-2 is specified, the national coded character set and collating sequence referenced are defined by the implementor. The implementor shall specify the ordinal number of each character for use when code-name-2 references a coded character set and the collating position of each character for use when code-name-2 references a collating sequence. The implementor also shall specify the correspondence between characters of the national coded character set specified by code-name-2 and the characters of the native national coded character set.

The coded character set referenced by code-name-2 is statically defined.

k) When literal-phrase is specified, the coded character set and collating sequence are defined according to the following rules, where the native coded character set is the type of coded character set or collating sequence being defined, either alphanumeric or national:

1. The value of each literal specifies:

   a. The ordinal number of a character within the native character set, if the literal is numeric. This value shall not exceed the value that represents the number of characters in the native character set.

   b. Otherwise, the actual character within the native character set. If the value of the literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.

2. The order in which the literals appear in the ALPHABET clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified.

3. Any characters of the native collating sequence that are not specified in the literal phrase shall assume a position in the collating sequence that is greater than that of the highest character specified in this literal phrase. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.

4. If a character code set is being specified, the implementor defines the ordinal number within the character code set being specified for each character within the native character set that is not specified by the literal-1 phrase.

5. If the THROUGH phrase is specified, the set of consecutive characters in the native character set beginning with the character specified by the value of literal-1, and ending with the character specified by the value of literal-2, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of consecutive characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

6. If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1 and literal-3 are assigned to the same ordinal position in the collating sequence being specified or in the character code set that is used to represent the data. Literal-1 is the first character in the sequence of multiple characters defined at that ordinal position. If alphabet-name-1 is referenced in a SYMBOLIC CHARACTERS clause, only literal-1 is used to represent the character in the native character set.

8) The character that has the highest ordinal position in the program collating sequence is associated with the figurative constant HIGH-VALUE, except when this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. If more than one character has the highest position in the program collating sequence, the last character specified is associated with the figurative constant HIGH-VALUE.

9) The character that has the lowest ordinal position in the program collating sequence is associated with the figurative constant LOW-VALUE, except when this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. If more than one character has the lowest position in the program collating sequence, the first character specified is associated with the figurative constant LOW-VALUE.

10) When specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with those characters having the highest and lowest positions, respectively, in the native national collating sequence, when the NATIONAL phrase is specified, or in the native alphanumeric collating sequence otherwise.

11) When the SYMBOLIC CHARACTERS clause is specified:

a) Symbolic-character-1 defines a figurative constant.

b) When ALPHANUMERIC is specified or implied, the value of figurative constant symbolic-character-1 is the representation of the coded character at ordinal position integer-1 in the native alphanumeric character set or, if the IN phrase is specified, in the character set referenced by alphabet-name-3.

c) When NATIONAL is specified, the value of figurative constant symbolic-character-1 is the internal representation of the character at ordinal position integer-1 in the native national character set or, if the IN phrase is specified, in the character set referenced by alphabet-name-3.

12) The CLASS clause provides a means for relating a name to the specified set of characters listed in that clause. The characters specified by the values of the literals in this clause define the exclusive set of characters of which class-name-1 consists.

The value of each literal specifies:

a) When the literal is numeric, the ordinal number of a character within the relevant native character set, or, when the IN phrase is specified, within the character set referenced by alphabet-name-4.

b) Otherwise, the actual character within the relevant native character set or, when the IN phrase is specified, within the character set referenced by alphabet-name-4. If the value of literal-5 contains multiple characters, each character in the literal is included in the set of characters identified by class-name-1.

If the THROUGH phrase is specified, the contiguous characters in the native character set beginning with the character specified by the value of literal-5, and ending with the character specified by the value of literal-6, are included in the set of characters identified by class-name-1. In addition, the contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

13) The CURRENCY SIGN clause is used to specify a currency string that is placed into numeric-edited data items when they are used as receiving items and de-edited from the data item when it is used as a sending item that has a numeric or numeric-edited receiving item. In addition, it is used to determine which symbol shall be used in a character-string of a basic format PICTURE clause to specify the presence of this currency string. This symbol is referred to as the currency symbol.

The runtime value ofliteral-7 is the currency string.

14) If the DECIMAL-POINT IS COMMA clause is specified, the functionality of the characters used to represent the decimal separator and the grouping separator are exchanged. The rules are as follows:

a) The character written in numeric literals to represent the decimal separator shall be the comma.

b) For the basic format of the PICTURE clause, the character written in character-strings, and inserted in numeric-edited items to represent the decimal separator shall be the comma.

NOTE 1 For the locale format of the PICTURE clause, the character written in character-strings to represent the decimal separator is always the period. The decimal separator to be inserted will be determined from the current locale.

The character written in character-strings, and inserted in numeric-edited items to represent the grouping separator shall be the period.

The DECIMAL-POINT IS COMMA clause has no effect on the editing or de-editing of a data item described with the locale format of the PICTURE clause.

NOTE 2 The DECIMAL-POINT IS COMMA clause is not processed until after the text manipulation stage of the compilation process and therefore does not impact literals specified in compiler directives, COPY statements, or REPLACE statements.

15) The content of the data item referenced by data-name-1 specifies the position of the cursor at the beginning of the execution of an ACCEPT screen statement. This content shall be updated by the execution of a successful ACCEPT screen statement to indicate the position of the visible cursor upon termination. (See 9.2.5, Cursor locator.)

16) Data-name-2 shall be updated during the execution of an ACCEPT screen statement as described in 9.2.3, CRT status.

17) When ORDER TABLE is specified, ordering-name-1 references an ordering table constructed in accordance with ISO/IEC 14651 and identified by literal-9. The implementor specifies the allowable content of literal-9.

18) If the PREFIXED phrase is specified in the ANY LENGTH STRUCTURE clause, data described with any-length-structure-name-1 is prefixed by a length field. If SIGNED is specified, the length field is a signed binary field; otherwise the length field is an unsigned binary field.

    The length field shall be capable of supporting the following values, at minimum:

ANY LENGTH STRUCTURE phrase	value
SIGNED PREFIXED	2147483647 [(2 ** 31) - 1]
PREFIXED	4294967295 [(2 ** 32) - 1]
SIGNED SHORT PREFIXED	32767 [(2 ** 15) - 1]
SHORT PREFIXED	65535 [(2 ** 16) - 1]

    If the implementor allows larger values, the maximum value that can be expressed in a length field associated with the SHORT PREFIXED phrase shall be expressible in a length field associated with the PREFIXED phrase.

    The maximum size of records described with an ANY LENGTH phrase is defined by the implementor.

19) If the DELIMITED phrase is specified in the ANY LENGTH STRUCTURE clause, a delimiter shall directly follow the data described with any-length-structure-name-1. The delimiter for alphanumeric any-length elementary items is X'00'. The delimiter for national any-length elementary items is NX'0000'.

12.3.7 REPOSITORY paragraph

The REPOSITORY paragraph allows specification of program prototype names, function prototype names, property-names, class names, and interface names that may be used within the scope of this environment division. It also allows declaration of intrinsic-function-names that may be used without specifying the word FUNCTION.

12.3.7.1 General format

```
REPOSITORY.
[ { class-specifier
  | interface-specifier
  | function-specifier
  | program-specifier
  | property-specifier } ... . ]
```

where class-specifier is:

```
CLASS class-name-1 [ AS literal-1 ] [ EXPANDS class-name-2 USING { class-name-3
                                                                 | interface-name-1 } ... ]
```

where interface-specifier is:

```
INTERFACE interface-name-2 [ AS literal-2 ] [ EXPANDS interface-name-3 USING { class-name-4
                                                                             | interface-name-4 } ... ]
```

where function-specifier is:

Format 1 (user-defined):

```
FUNCTION function-prototype-name-1 [ AS literal-3 ]
```

Format 2 (intrinsic):

```
FUNCTION { { intrinsic-function-name-1 } ...
         | ALL                             } INTRINSIC
```

where program-specifier is:

```
PROGRAM program-prototype-name-1 [ AS literal-4 ]
```

where property-specifier is:

```
PROPERTY property-name-1 [ AS literal-5 ]
```

12.3.7.2 Syntax rules

ALL SPECIFIERS

1) If any class-name-1, interface-name-2, program-prototype-name-1, function-prototype-name-1, intrinsic-function-name-1 or property-name-1 is specified more than once in the REPOSITORY paragraph, all the specifications for that name shall be identical.

2) Literal-1, literal-2, literal-3, literal-4, and literal-5 shall be alphanumeric literals or national literals and shall be neither figurative constants nor zero-length literals.

3) The EXPANDS phrase shall not be specified in the REPOSITORY paragraph of a class definition that contains a USING clause in its CLASS-ID paragraph or of an interface definition that contains a USING clause in its INTERFACE-ID paragraph.

CLASS SPECIFIER

4) Class-name-2, class-name-3, and interface-name-1 shall be defined in the same REPOSITORY paragraph where class-name-1 is defined.

5) If the specified class-name-1 is the name of the class definition in which this REPOSITORY paragraph is specified, references to class-name-1 are to that class definition and this class-specifier is ignored.

6) If the CLASS phrase is specified without the EXPANDS phrase:

   a) If literal-1 is specified, there shall be information in the external repository for the class literal-1.

   b) If literal-1 is not specified, there shall be information in the external repository for the class class-name-1.

INTERFACE SPECIFIER

7) Interface-name-3, class-name-4, and interface-name-4 shall be defined in the same REPOSITORY paragraph where interface-name-2 is defined.

8) If the specified interface-name-2 is the name of the interface definition in which this REPOSITORY paragraph is specified, references to interface-name-2 are to that interface definition and this interface-specifier is ignored.

9) If the INTERFACE phrase is specified without the EXPANDS phrase:

   a) If literal-2 is specified, there shall be information in the external repository for the interface literal-2.

   b) If literal-2 is not specified, there shall be information in the external repository for the interface interface-name-2.

FUNCTION SPECIFIER

10) Literal-3, if specified, or function-prototype-name-1, if literal-3 is not specified, shall be one of the following:

   — the name of a function prototype specified in this compilation group
   — the name of a function definition specified previously in this compilation group
   — the name of a function for which information exists in the external repository.

11) If the specified function-prototype-name-1 is the name of the function definition in which this REPOSITORY paragraph is specified, references to function-prototype-name-1 are to that function definition and this function-specifier is ignored.

12) Intrinsic-function-name-1 shall not be specified as a user-defined word within the scope of this REPOSITORY paragraph.

13) If ALL is specified in the intrinsic format of the function-specifier, none of the names of the intrinsic functions may be specified as a user-defined word within the scope of this REPOSITORY paragraph.

PROGRAM SPECIFIER

14) Literal-4, if specified, or program-prototype-name-1, if literal-4 is not specified, shall be one of the following:

— the name of a program prototype specified in this compilation group
— the name of a program definition specified previously in this compilation group
— the name of a program for which information exists in the external repository.

15) If the specified program-prototype-name-1 is the name of the program definition in which this REPOSITORY paragraph is specified or the name of a containing program definition, references to program-prototype-name-1 are to the named program definition and this program-specifier is ignored.

PROPERTY SPECIFIER

16) If the PROPERTY phrase is specified,

a) If literal-5 is specified, there shall be information in the external repository for the property literal-5 that is part of one of the classes or interfaces that are declared in this REPOSITORY paragraph.

b) If literal-5 is not specified, there shall be information in the external repository for the property property-name-1 that is part of one of the classes or interfaces that are declared in this REPOSITORY paragraph.

12.3.7.3 General rules

1) Class-name-1 is the name of a class that may be used throughout the scope of the containing environment division.

If class-name-1 is a class described with the USING phrase, class-name-1 may be specified only in the REPOSITORY paragraph.

2) If the AS phrase is specified, literal-1, literal-2, literal-3, or literal-4 is the externalized name by which the class, interface, function, or program, respectively, is known to the operating environment. Literal-5 is the externalized name known to the operating environment for a method that implements the named property. The implementor shall specify when the AS phrase is required.

3) Class-name-3 and interface-name-1 are actual parameters for the parameterized class referenced by class-name-2.

4) Class-name-4 and interface-name-4 are actual parameters for the parameterized interface referenced by interface-name-3.

5) If the EXPANDS phrase is specified in a class-specifier, a class class-name-1 is created from the parameterized class class-name-2. The number of parameters in the USING phrase of the EXPANDS phrase of the class-specifier shall be the same as the number of parameters in the USING clause of the CLASS-ID paragraph of class-name-2. The interface for class-name-1 is the interface specified for class-name-2 with the parameters of class-name-2 replaced by the parameters specified in the class-specifier.

The class class-name-1 is created from the parameterized class class-name-2 by replacing each specification of the formal parameter by the corresponding actual parameter.

6) The compiler shall use the information specified for class-name-1 together with the external repository to determine the details of the class that is to be used. It is implementor-defined how the information in the class specifier and the external repository are used to determine which class is used.

7) Interface-name-2 is the name of an interface that may be used throughout the scope of the containing environment division.

   If interface-name-1 is an interface described with the USING phrase, interface-name-1 may be specified only in the REPOSITORY paragraph.

8) If the EXPANDS phrase is specified in an interface-specifier, an interface interface-name-2 is created from the parameterized interface interface-name-3. The number of parameters in the USING phrase of the EXPANDS phrase of the interface-specifier shall be the same as the number of parameters in the USING phrase of the INTERFACE-ID paragraph of interface-name-3. The interface for interface-name-2 is the interface specified for interface-name-3 with the parameters of interface-name-3 replaced by the parameters specified in the interface-specifier.

   The interface interface-name-2 is created from the parameterized interface interface-name-3 by replacing each specification of the formal parameter by the corresponding actual parameter.

9) The compiler shall use the information specified for interface-name-2 together with the external repository to determine the details of the interface that is to be used. It is implementor-defined how the information in the interface specifier and the external repository are used to determine which interface is used.

10) Program-prototype-name-1 is the name of a program prototype that may be used throughout the scope of the containing environment division. The details for calling a program via this program prototype are obtained as follows:

    a) if the externalized name of the program prototype is the externalized name of a program definition specified previously in the same compilation group, the details are taken from that program definition, which is the program that will be called, and the details in the external repository are ignored; otherwise,

       NOTE Literal-4, if specified, is the externalized name of the program prototype; otherwise, the externalized name is program-prototype-name-1.

    b) if the externalized name of the program prototype is the externalized name of a program prototype definition specified in the same compilation group, the details are taken from that program prototype definition and the details in the external repository are ignored. The program that will be called is the one with the same externalized name as the externalized name of the program prototype; otherwise,

    c) the details are taken from the external repository for the program with the same name as the externalized name of the program prototype. That program is the one that will be called.

11) Function-prototype-name-1 is the name of a function prototype that may be used throughout the scope of the containing environment division. The details for activating a function via this function prototype are obtained as follows:

    a) if the externalized name of the function prototype is the externalized name of a function definition specified previously in the same compilation group, the details are taken from that function definition, which is the function that will be activated, and the details in the external repository are ignored; otherwise,

       NOTE Literal-3, if specified, is the externalized name of the function prototype; otherwise, the externalized name is function-prototype-name-1.

    b) if the externalized name of the function prototype is the externalized name of a function prototype definition specified in the same compilation group, the details are taken from that function prototype definition and the details in the external repository are ignored. The function that will be activated is the one with the same externalized name as the externalized name of the function prototype; otherwise,

c) the details are taken from the external repository for the function with the same name as the externalized name of the function prototype. That function is the one that will be activated.

12) Within the scope of the containing environment division, a reference to function-prototype-name-1 is a reference to a user-defined function, and not to an intrinsic function of the same name.

13) Within the scope of the containing environment division, intrinsic-function-name-1 may be specified as a function-identifier without being preceded by the keyword FUNCTION, unless specific rules require the use of the keyword FUNCTION.

14) If ALL is specified in the intrinsic format of the function-specifier, it is as if each of the intrinsic-function-names defined in 8.11, Intrinsic function names, were specified.

15) Property-name-1 is the name of an object property that may be used throughout the scope of the containing environment division.

12.4 Input-output section

The input-output section deals with the information needed to control transmission and handling of data between external media and a runtime element.

12.4.1 General format

```
INPUT-OUTPUT SECTION.
[ file-control-paragraph ]
[ i-o-control-paragraph ]
```

12.4.2 Syntax rules

1) The input-output section may be specified in a program definition or a function definition. Within a class definition, the input-output section may be specified only in a factory definition or instance definition, but not in a method definition. The input-output section shall not be specified within an interface definition.

12.4.3 FILE-CONTROL paragraph

The FILE-CONTROL paragraph specifies file-related information.

12.4.3.1 General format

```
FILE-CONTROL. [ file-control-entry ] ...
```

12.4.4 File control entry

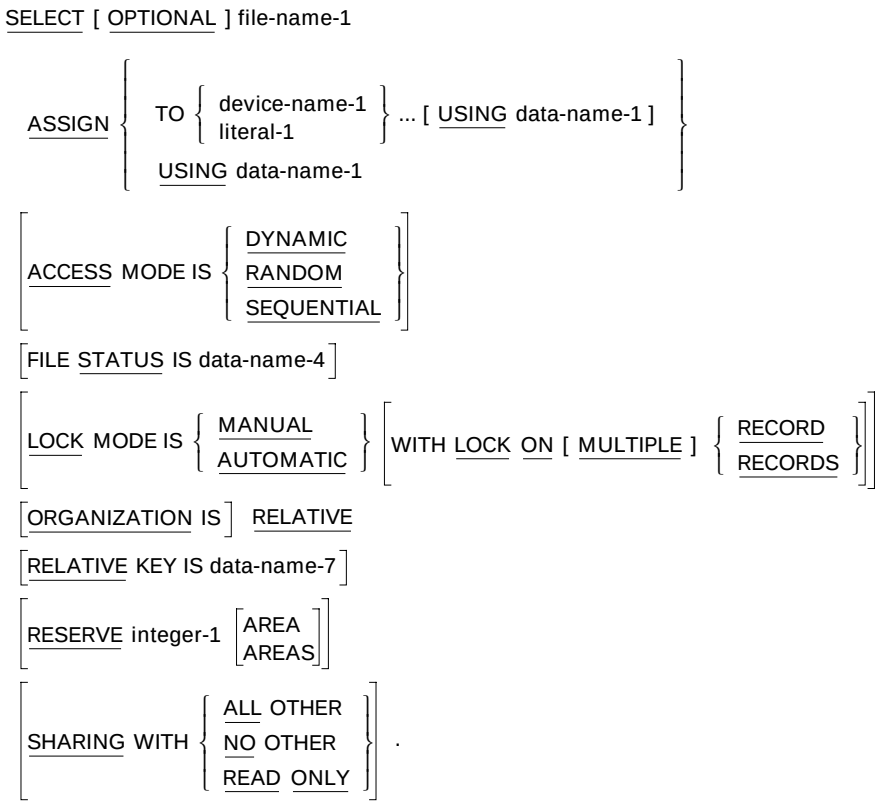
The file control entry declares the relevant physical attributes of a file.

12.4.4.1 General format

Format 1 (indexed):

```
SELECT [ OPTIONAL ] file-name-1

  ASSIGN { TO { device-name-1 } ... [ USING data-name-1 ] }
         { literal-1          }
         { USING data-name-1                              }

[ ACCESS MODE IS { DYNAMIC    } ]
                 { RANDOM     }
                 { SEQUENTIAL }

[ ALTERNATE RECORD KEY IS { data-name-2                                    } [ WITH DUPLICATES ] ]...
                          { record-key-name-1 SOURCE IS { data-name-3 } ... }

[ collating-sequence-clause ]...

[ FILE STATUS IS data-name-4 ]

[ LOCK MODE IS { MANUAL    } [ WITH LOCK ON [ MULTIPLE ] { RECORD  } ] ]
               { AUTOMATIC }                             { RECORDS }

[ ORGANIZATION IS ] INDEXED

  RECORD KEY IS { data-name-5                                    }
                { record-key-name-2 SOURCE IS { data-name-6 } ... }

[ RESERVE integer-1 [ AREA  ] ]
                    [ AREAS ]

[ SHARING WITH { ALL OTHER } ]
               { NO OTHER  }
               { READ ONLY }
.
```

Format 2 (relative):



Format 3 (sequential):

```
SELECT [ OPTIONAL ] file-name-1

    ASSIGN { TO { device-name-1 } ... [ USING data-name-1 ] }
           { USING data-name-1 }

  [ ACCESS MODE IS SEQUENTIAL ]
  [ FILE STATUS IS data-name-4 ]
  [ LOCK MODE IS { MANUAL    } [ WITH LOCK ON { RECORD  } ] ]
                 { AUTOMATIC }                { RECORDS }
  [ [ ORGANIZATION IS ] SEQUENTIAL ]
  [ RECORD DELIMITER IS { STANDARD-1  } ]
                        { feature-name-1 }
  [ RESERVE integer-1 [ AREA  ] ]
                      [ AREAS ]
  [ SHARING WITH { ALL OTHER } ]
                 { NO OTHER  }
                 { READ ONLY }
```

Format 4 (sort-merge):

```
SELECT [ OPTIONAL ] file-name-1

    ASSIGN { TO { device-name-1 } ... [ USING data-name-1 ] }
           { USING data-name-1 }

  [ [ ORGANIZATION IS ] SEQUENTIAL ] .
```

where collating-sequence-clause is described in 12.4.4.6, COLLATING SEQUENCE clause.

12.4.4.2 Syntax rules

ALL FORMATS

1) The SELECT clause shall be specified first in the file control entry. The clauses that follow the SELECT clause may appear in any order.

2) A given file-name may be specified in only one SELECT clause within a factory, function, object, or program.

3) For each file-name specified in a SELECT clause, there shall be a file description entry or a sort-merge file description entry in the file section of the factory, function, object, or program in which the SELECT clause is specified.

4) Literal-1 shall be an alphanumeric literal and shall be neither a figurative constant nor a zero-lenght literal.

5) The meaning and rules for the allowable specification of device-name-1 and the value of literal-1 are defined by the implementor.

6) The allowable and required syntactical combinations of data-name-1 with device-name-1 and of data-name-1 with literal-1 are defined by the implementor.

7) Data-name-1 shall reference an alphanumeric data item and shall not be subordinate to the file description entry for file-name-1.

8) Data-name-1 may be qualified.

FORMAT 1

9) Format 1 shall be specified only for an indexed file. The associated file description entry shall not be a sort-merge file description entry.

FORMAT 2

10) Format 2 shall be specified only for a relative file. The associated file description entry shall not be a sort-merge file description entry.

11) The RELATIVE clause shall be specified if the DYNAMIC or RANDOM phrase of the ACCESS clause is specified.

FORMAT 3

12) Format 3 shall be specified only for a sequential file or a report file. The associated file description entry shall not be a sort-merge file description entry.

FORMAT 4

13) Format 4 shall be specified only for a sort-merge file. The associated file description entry shall be a sort-merge file description entry.

12.4.4.3 General rules

ALL FORMATS

1) If the file connector referenced by file-name-1 is an external file connector (see 13.18.21, EXTERNAL clause), all file control entries in the run unit that reference this file connector shall have:

   a) The same specification for the OPTIONAL phrase.

   b) A consistent specification for data-name-1, device-name-1, and literal-1 in the ASSIGN clause. The implementor shall specify the consistency rules for data-name-1, device-name-1, and literal-1.

   c) Either the STANDARD-1 phrase or a consistent value of feature-name-1 in the RECORD DELIMITER clause. The implementor shall specify the consistency rules for feature-name-1.

   d) The same value for integer-1 in the RESERVE clause.

   e) The same organization.

   f) The same access mode.

   g) The same specification of COLLATING SEQUENCE clauses.

h) The same specification of the RELATIVE KEY clause, where data-name-7 references an external data item.

i) The same data description entry for data-name-5 and each data-name-6 as well as their relative location within the associated record.

j) The same data description entry for data-name-2 and each data-name-3 as well as their relative location within the associated record, the same number of alternate record keys, and the same DUPLICATES phrase.

k) The same sharing mode.

l) The same lock mode and the same choice of either single record locking or multiple record locking.

2) The OPTIONAL phrase applies only to files opened in the input, I-O, or extend mode. Its specification is required for physical files that are not necessarily present each time the runtime element is executed.

3) The ASSIGN clause specifies the association of the file connector referenced by file-name-1 to a physical file identified by device-name-1, literal-1, or the content of the data item referenced by data-name-1. The association occurs at the time of execution of an OPEN, SORT, or MERGE statement that referenced file-name-1, according to the following rules:

a) When the TO phrase of the ASSIGN clause is specified and the USING phrase is omitted, the file connector referenced by file-name-1 is associated with a physical file identified by the specification of device-name-1 or the value of literal-1 in the source unit that specifies the OPEN, SORT, or MERGE statement.

b) When the USING phrase of the ASSIGN clause is specified, the file connector referenced by file-name-1 is associated with a physical file identified by the content of the data item referenced by data-name-1 in the runtime element that executes the OPEN, SORT, or MERGE statement.

If the association cannot be made because the content of the data item referenced by data-name-1 is not consistent with the specification for device-name-1 or literal-1, the OPEN, SORT, or MERGE statement is unsuccessful.

4) When the USING phrase is specified, the meaning and rules for the allowable content of the data item referenced by data-name-1 are defined by the implementor. The consistency rules, if any, that apply between the content of the data item referenced by data-name-1 and either the specification of device-name-1 or the value of literal-1 are defined by the implementor.

FORMAT 1

5) The indexed format defines a file connector for an indexed file.

6) If no COLLATING SEQUENCE clause is specified:

a) the collating sequence for alphanumeric record keys, both primary and alternate, is the native alphanumeric collating sequence;

b) the collating sequence for national record keys, both primary and alternate, is the native national collating sequence.

FORMAT 2

7) The relative format defines a file connector for a relative file.

FORMAT 3

8) The sequential format defines a file connector for a sequential file.

FORMAT 4

9) The sort-merge format defines a file connector for a sort-merge file.

12.4.4.4 ACCESS MODE clause

The ACCESS MODE clause specifies the order in which records are to be accessed in the file.

12.4.4.4.1 General format

```
ACCESS MODE IS { SEQUENTIAL }
               { RANDOM     }
               { DYNAMIC    }
```

12.4.4.4.2 Syntax rules

1) The RANDOM clause shall not be specified for file-names specified in the USING or GIVING phrase of a SORT or MERGE statement.

2) The DYNAMIC and RANDOM phrases shall not be specified for a sequential file.

12.4.4.4.3 General rules

1) If the ACCESS MODE clause is not specified, sequential access is assumed.

2) If the access mode is sequential, records in the file are accessed in the sequence dictated by the file organization:

   a) For sequential files this sequence is specified by predecessor-successor record relationships established by the execution of WRITE statements when the physical file is created or extended.

   b) For relative files this sequence is the order of ascending relative record numbers of existing records in the physical file.

   c) For indexed files this sequence is ascending within a given key of reference according to the collating sequence for that key.

3) If the access mode is random:

   a) For a relative file, the value of a relative key data item indicates the record to be accessed.

   b) For an indexed file, the value of a record key data item indicates the record to be accessed.

4) If the access mode is dynamic, records in the file may be accessed sequentially, randomly, or both.

12.4.4.5 ALTERNATE RECORD KEY clause

The ALTERNATE RECORD KEY clause specifies an alternate record key access path to the records in an indexed file.

12.4.4.5.1 General format

```
ALTERNATE RECORD KEY IS { data-name-1                                          } [ WITH DUPLICATES ]
                        { record-key-name-1 SOURCE IS { data-name-2 } ... }
```

12.4.4.5.2 Syntax rules

1) Data-name-1 and data-name-2 may be qualified.

2) Data-name-1 and data-name-2 shall be defined as a data item of category alphanumeric or national within a record description entry associated with the file-name to which the ALTERNATE RECORD KEY clause is subordinate. All occurrences of data-name-2 shall be of the same category.

3) Data-name-1 and data-name-2 shall not reference a variable-length data item.

4) Data-name-1 shall not reference an item whose leftmost byte position corresponds to the leftmost byte position of the prime record key, or of another alternate record key. This restriction does not apply in the case where either key is specified using the SOURCE phrase.

5) If the indexed file contains variable-length records, each data-name-1 and data-name-2 shall be contained within the first x bytes of the record, where x equals the minimum record size specified for the file. (See 13.18.42, RECORD clause.)

6) Record-key-name-1 has the class and category of data-name-2.

12.4.4.5.3 General rules

1) An ALTERNATE RECORD KEY clause specifies an alternate record key for the file with which this clause is associated.

2) Record-key-name-1 defines a record key consisting of the concatenation of all occurrences of data-name-2 in the order specified.

3) The data description of data-name-1 or data-name-2 as well as their relative location within a record shall be the same as that used when the physical file was created. The number of alternate record keys for the file shall also be the same as that used when the physical file was created.

4) The DUPLICATES phrase specifies that the value of the associated alternate record key may be equal to the value of the same alternate record key in another record in the physical file. If the DUPLICATES phrase is not specified, the value of the associated alternate record key shall not be equal to the value of the same alternate record key in another record in the physical file. The equality or inequality is based on the collating sequence used for the file according to the rules for a relation condition.

5) The identical byte positions referenced by data-name-1 or data-name-2 in any one record description entry are implicitly referenced as keys for all other record description entries of that file.

12.4.4.6 COLLATING SEQUENCE clause

The COLLATING SEQUENCE clause specifies the collating sequence to be used for the ordering of record keys and alternate record keys for an indexed file. Multiple collating sequences can be used by specifying a collating sequence clause unique to the primary key or to specific alternate record keys.

12.4.4.6.1 General format

Format 1 (file-level)

```
COLLATING SEQUENCE { IS alphabet-name-1 [ alphabet-name-2 ]
                   { [ FOR ALPHANUMERIC IS alphabet-name-1 ]
                   { [ FOR NATIONAL IS alphabet-name-2 ]
```

Format 2 (key-level)

```
COLLATING SEQUENCE OF { data-name-1 }
                      { record-key-name-1 } ... IS alphabet-name-3
```

12.4.4.6.2 Syntax rules

FORMAT 1

1) Alphabet-name-1 shall reference an alphabet that defines an alphanumeric collating sequence.

2) Alphabet-name-2 shall reference an alphabet that defines a national collating sequence.

3) Only one file-level format COLLATING SEQUENCE clause may be specified in one file control entry.

FORMAT 2

4) Data-name-1 shall be a name specified as the data-name in an ALTERNATE RECORD KEY clause or in a RECORD KEY clause in the file control entry.

5) Record-key-name-1 shall be a name specified as a record-key-name in an ALTERNATE RECORD KEY clause or in a RECORD KEY clause in the file control entry.

6) When the class of data-name-1 or record-key-name-1 is national, alphabet-name-3 shall reference an alphabet that defines a national collating sequence; otherwise, alphabet-name-3 shall reference an alphabet that defines an alphanumeric collating sequence.

7) Neither data-name-1 nor record-key-name-1 shall be specified in more than one COLLATING SEQUENCE clause.

12.4.4.6.3 General rules

ALL FORMATS

1) Each collating sequence is a fixed file attribute and the collating sequence is set upon the successful execution of an OPEN statement that creates the physical file. The collating sequence used is the one that applies as specified in the following general rules.

FORMAT 1

2) An alphanumeric collating sequence referenced by alphabet-name-1 applies to any record keys of class alphanumeric, whether primary or alternate, not specified in a key-level format of another COLLATING SEQUENCE clause in the file control entry.

3) A national collating sequence referenced by alphabet-name-2 applies to any record keys of class national, whether primary or alternate, not specified in a key-level format of another COLLATING SEQUENCE clause in the file control entry.

4) If alphabet-name-1 is not specified in the file control entry, the native alphanumeric collating sequence applies to any record keys of class alphanumeric, whether primary or alternate, not specified in a key-level format of another COLLATING SEQUENCE clause.

5) If alphabet-name-2 is not specified in the file control entry, the native national collating sequence applies to any record keys of class national, whether primary or alternate, not specified in a key-level format of another COLLATING SEQUENCE clause.

FORMAT 2

6) Alphabet-name-3 applies to record keys identified by data-name-1 or record-key-name-1.

12.4.4.7 FILE STATUS clause

The FILE STATUS clause specifies a data item that contains the status of an input-output operation.

12.4.4.7.1 General format

FILE STATUS IS data-name-1

12.4.4.7.2 Syntax rules

1) Data-name-1 may be qualified.

2) Data-name-1 shall be a two-character data item of the category alphanumeric, defined in the working-storage, local-storage, or linkage section.

3) Data-name-1 shall not be an any-length elementary item or a variable-length group.

12.4.4.7.3 General rules

1) If the FILE STATUS clause is specified, the data item referenced by data-name-1 is updated to contain the value of the I-O status for the file connector referenced by the subject of the entry when the I-O status associated with that file connector is updated as a result of an input-output statement.

NOTE In the case where a file-name is global and data-name-1 is not, data-name-1 is updated by references to file-name in contained programs even though data-name-1 is a local name.

12.4.4.8 LOCK MODE clause

The LOCK MODE clause indicates the type of record locking for a shared file.

12.4.4.8.1 General format

```
LOCK MODE IS { MANUAL    } [ WITH LOCK ON [ MULTIPLE ] { RECORD  } ]
             { AUTOMATIC }                             { RECORDS }
```

12.4.4.8.2 Syntax rules

1) The MULTIPLE phrase shall not be specified for a file described with sequential organization or sequential access mode.

12.4.4.8.3 General rules

1) If the LOCK MODE clause is omitted from a file control entry,

   a) If there is a SHARING clause in that file control entry, no record locks are set by the execution of I-O statements through the associated file connector.

   b) If there is no SHARING clause in that file control entry,

      1. If an OPEN statement for the associated file connector has a SHARING phrase, no record locks are set by the execution of I-O statements for that opening of the associated file connector.

      2. If an OPEN statement for the associated file connector has no SHARING phrase, the type of record locking for that opening of a shared file associated with that file connector is defined by the implementor. The implementor may define the default in terms of standard LOCK MODE clause syntax, specify another type of record locking as the default, or specify that the default is no record locking. If the default is defined in terms of standard syntax, then the semantics shall be as if that clause had been specified in the file control entry.

2) If the processor does not support record locking, record locks have no effect for the associated file connector.

3) If a physical file is open in the sharing with no other mode, the LOCK MODE clause has no effect. Otherwise, the LOCK MODE clause has the effects described in the general rules that follow.

4) If the AUTOMATIC phrase is specified, the lock mode is automatic. Records are locked when any READ statement is executed.

5) If the MANUAL phrase is specified, the lock mode is manual. Records locks are obtained only when the LOCK phrase is explicitly specified on an I-O statement.

6) Single record locking is specified explicitly by the LOCK ON phrase without the MULTIPLE phrase and implicitly when the LOCK MODE clause is specified with the LOCK ON phrase omitted. Single record locking allows only one record of a file to be locked at a given time through a single file connector. Execution of any I-O statement except START releases any previously locked record in that file for that file connector. Details of record locking are specified in 9.1.15, Record locking.

7) If the MULTIPLE phrase is specified in the LOCK ON phrase, then multiple record locking is said to have been specified and a file connector is permitted to have more than one record of a file locked. A file connector that has specified multiple record locking for a file may hold a number of record locks for that file simultaneously. This prevents other file connectors from accessing any member of the set of locked records, but will not deny them access to records that are not locked. The implementor shall specify the maximum number of record locks that may be held by a file connector; that maximum shall be at least 15. The implementor shall specify

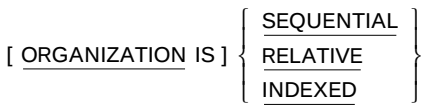
the maximum number of record locks that may be held by a run unit; that maximum shall be at least 255. Any I-O statement that attempts to obtain a record lock that would exceed either limit is unsuccessful and receives an I-O status that indicates that condition. Details of record locking are specified in 9.1.15, Record locking.

8) The setting of a record lock is part of the atomic operation of an I-O statement.

12.4.4.9 ORGANIZATION clause

The ORGANIZATION clause specifies the logical structure of a file.

12.4.4.9.1 General format



12.4.4.9.2 General rules

1) The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a physical file is created and cannot subsequently be changed.

2) The SEQUENTIAL phrase specifies that the file organization is sequential. Sequential organization is a permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

3) The RELATIVE phrase specifies that the file organization is relative. Relative organization is a permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, that specifies the record's logical ordinal position in the file.

4) The INDEXED phrase specifies that the file organization is indexed. Indexed organization is a permanent logical file structure in which each record is identified by the value of one or more keys within that record.

5) When the ORGANIZATION clause is not specified, sequential organization is implied.

12.4.4.10 RECORD DELIMITER clause

The RECORD DELIMITER clause indicates the method of determining the length of a variable-length record on the external medium.

12.4.4.10.1 General format

```
RECORD DELIMITER IS { STANDARD-1    }
                    { feature-name-1 }
```

12.4.4.10.2 Syntax rules

1) The RECORD DELIMITER clause may be specified only for variable-length records.

NOTE There are three ways variable-length records may be specified:

a) The RECORD clause is not specified and the implementor has specified that variable-length records are obtained in this circumstance.

b) The RECORD IS VARYING clause is specified.

c) The format 3 RECORD CONTAINS clause is specified and the implementor has specified that variable-length records are obtained in this circumstance.

2) The implementor shall specify the names available for use as feature-name-1.

12.4.4.10.3 General rules

1) The RECORD DELIMITER clause indicates the method of determining the length of a variable-length record on the external medium. Any method used shall not be reflected in the record area or the record size used within the function, method, or program.

2) If STANDARD-1 is specified, the external medium shall be a magnetic tape file.

3) If STANDARD-1 is specified, the method used for determining the length of a variable-length record is that specified in ISO 1001:1986, 7.2, Records.

4) If feature-name-1 is specified, the method used for determining the length of a variable-length record is that associated with feature-name-1 by the implementor.

5) If the RECORD DELIMITER clause is not specified, the method used for determining the length of a variable-length record is specified by the implementor.

6) At the time of a successful execution of an OPEN statement, the record delimiter is the one specified in the RECORD DELIMITER clause in the file control entry associated with the file-name specified in the OPEN statement.

12.4.4.11 RECORD KEY clause

The RECORD KEY clause specifies the prime record key access path to the records in an indexed file.

12.4.4.11.1 General format

```
RECORD KEY IS { data-name-1                                          }
              { record-key-name-1 SOURCE IS { data-name-2 } ...      }
```

12.4.4.11.2 Syntax rules

1) Data-name-1 and data-name-2 may be qualified.

2) Data-name-1 and data-name-2 shall reference a data item of category alphanumeric or category national within a record description entry associated with the file-name specified in this file control entry. All occurrences of data-name-2 shall be of the same category.

3) Data-name-1 and data-name-2 shall not reference a variable-length data item.

4) If the indexed file contains variable-length records, data-name-1 and each data-name-2 shall be contained within the first n bytes of the record, where n equals the minimum record size specified for the file. (See 13.18.42, RECORD clause.)

5) Record-key-name-1 has the class and category of data-name-2.

12.4.4.11.3 General rules

1) The RECORD KEY clause specifies the prime record key for the file that is the subject of the entry. The value of the associated record key shall not be equal to the value of the same record key in another record in the file. The equality or inequality is based on the collating sequence used for the file according to the rules for a relation condition.

2) Record-key-name-1 defines a record key consisting of the concatenation of all occurrences of data-name-2 in the order specified.

3) The data description of data-name-1 or data-name-2 as well as their relative location within a record shall be the same as that used when the file was created.

4) If the file has more than one record description entry, data-name-1 or data-name-2 need only be described in one of these record description entries. The identical byte positions referenced by data-name-1 or data-name-2 in any one record description entry are implicitly referenced as keys for all other record description entries of that file.

12.4.4.12 RELATIVE KEY clause

The RELATIVE KEY clause identifies the data item that will contain the relative record number for accessing a relative file.

12.4.4.12.1 General format

RELATIVE KEY IS data-name-1

12.4.4.12.2 Syntax rules

1) Data-name-1 may be qualified.

2) Data-name-1 shall reference an unsigned integer data item whose description does not contain the picture symbol 'P'.

3) Data-name-1 shall not be defined in a record description entry subordinate to the associated file-name.

12.4.4.12.3 General rules

1) All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4, ...

2) The relative key data item associated with the execution of an input-output statement is the data item referenced by data-name-1; data-name-1 is used to communicate a relative record number between the user and the mass storage control system (MSCS).

12.4.4.13 RESERVE clause

The RESERVE clause allows the user to specify the number of input-output areas allocated.

12.4.4.13.1 General format

```
RESERVE integer-1 [ AREA  ]
                  [ AREAS ]
```

12.4.4.13.2 General rules

1) If the RESERVE clause is specified, the number of input-output areas allocated is equal to the value of integer-1. If the RESERVE clause is not specified, the number of input-output areas allocated is specified by the implementor.

12.4.4.14 SHARING clause

The SHARING clause indicates that a file is to participate in file sharing and record locking. It specifies the degree of file sharing (or non-sharing) to be permitted for a file and whether record locks have an effect.

12.4.4.14.1 General format

```
SHARING WITH { ALL OTHER  }
             { NO OTHER   }
             { READ ONLY  }
```

12.4.4.14.2 General rules

1) The SHARING clause specifies the sharing mode to be used for the file unless it is overridden by the SHARING phrase of the OPEN statement. This clause also specifies whether record locks have an effect. Additional details are specified in 9.1.14, Sharing mode.

12.4.5 I-O-CONTROL paragraph

The I-O-CONTROL paragraph specifies that memory areas associated with different files are to be shared during file processing, record processing, or sort-merge processing.

12.4.5.1 General format

I-O-CONTROL. [[{ same-clause } ...] .]

where same-clause is shown below

12.4.6 SAME clause

The SAME clause specifies files for which memory areas are to be shared during file processing, record processing, or sort-merge processing.

12.4.6.1 General format

Format 1 (file-area):

SAME AREA FOR file-name-1 { file-name-2 } ...

Format 2 (record-area):

SAME RECORD AREA FOR file-name-1 { file-name-2 } ...

Format 3 (sort-merge-area):

SAME { SORT / SORT-MERGE } AREA FOR file-name-1 { file-name-2 } ...

12.4.6.2 Syntax rules

1) SORT and SORT-MERGE are equivalent.

2) File-name-1 and file-name-2 shall be specified in the FILE-CONTROL paragraph of the source element that contains this SAME clause.

3) File-name-1 and file-name-2 shall not reference an external file connector.

4) The files specified in a given SAME clause need not all have the same organization or access.

5) A given file-name that represents a report file may be specified in one file-area format SAME clause and shall not be specified in a record-area format or sort-merge-area format SAME clause.

6) A given file-name that represents a sort or merge file may be specified in one record-area format SAME clause and in one sort-merge-area SAME clause, and shall not be specified in a file-area format SAME clause.

7) A given file-name that represents a file other than a report file or a sort or merge file may be specified in one file-area format, in one record-area format, and in one or more sort-merge-area format SAME clauses.

8) At least one file-name specified in a sort-merge-area format SAME clause shall represent a sort or merge file.

9) If one or more file-names specified in a file-area format SAME clause are also specified in a record-area format SAME clause, all of the file-names specified in the file-area format SAME clause shall also be specified in the

record-area format SAME clause. Additional file-names not specified in the file-area format SAME clause may be specified in the record-area format SAME clause.

10) If a file-name that represents a file other than a sort or merge file is specified in a file-area format SAME clause and in one or more sort-merge-area format SAME clauses, all of the file-names specified in that file-area format SAME clause shall also be specified in those sort-merge-area format SAME clause(s).

12.4.6.3 General rules

1) A file-area format SAME clause specifies that two or more files referenced by file-name-1, file-name-2 are to use the same memory area during processing. The area being shared includes all storage areas assigned to the files referenced by file-name-1, file-name-2. No more than one of these files may be in the open mode at a given time.

2) A record-area format SAME clause specifies that two or more files referenced by file-name-1, file-name-2 are to share a memory area for processing the current logical record. All of these files may be in the open mode at the same time, except that only one file that is also specified in a file-area format SAME clause may be open at that time. A logical record in the shared memory area is a logical record of each file open in the output mode and of the most recently-read file open in the input mode. This is equivalent to an implicit redefinition of the area with records aligned on the leftmost byte position. The record area is available to the runtime element when any file connector referenced by file-name-1, file-name-2, ... is open. When none of the file connectors is open, the record area is not available to the runtime element.

3) A sort-merge-area format SAME clause specifies that memory is shared as follows:

   a) Any storage area allocated for the sorting or merging of a sort or merge file specified in a sort-merge-area format SAME clause is available for reuse in sorting or merging any of the other sort or merge files specified in that sort-merge-area format SAME clause.

   b) Storage areas assigned to files specified in a sort-merge-area format SAME clause that do not represent sort or merge files may be allocated as needed for sorting or merging the sort or merge files named in that sort-merge-area format SAME clause. The extent of such allocation shall be specified by the implementor.

   c) Storage areas assigned to files specified in a sort-merge-area format SAME clause other than sort or merge files do not share the same storage area with each other.

4) During the processing of a SORT or MERGE statement that refers to a sort or merge file named in a sort-merge-area format SAME clause, any non-sort and non-merge files associated with file-names specified in that clause shall not be in the open mode.

13 Data division

13.1 General

The data division describes the data that the runtime module is to accept as input, to manipulate, to create, or to produce as output. The data division is optional.

The following is the general format of the sections in the data division and defines the order of their presentation in the source element.

13.2 Data division structure

13.2.1 General format

```
DATA DIVISION.
[ FILE SECTION. [ file-description-entry [ constant-entry
                                           record-description-entry
                                           type-declaration-entry ] ...
                  sort-merge-file-description-entry { constant-entry
                                                      record-description-entry
                                                      type-declaration-entry } ... ] ... ]
[ WORKING-STORAGE SECTION. [ 77-level-description-entry
                             constant-entry
                             record-description-entry
                             type-declaration-entry ] ... ]
[ LOCAL-STORAGE SECTION. [ 77-level-description-entry
                           constant-entry
                           record-description-entry
                           type-declaration-entry ] ... ]
[ LINKAGE SECTION. [ 77-level-description-entry
                     constant-entry
                     record-description-entry
                     type-declaration-entry ] ... ]
[ REPORT SECTION. [ report-description-entry { constant-entry
                                               report-group-description-entry } ... ] ... ]
[ SCREEN SECTION. [ constant-entry
                    screen-description-entry ] ... ]
```

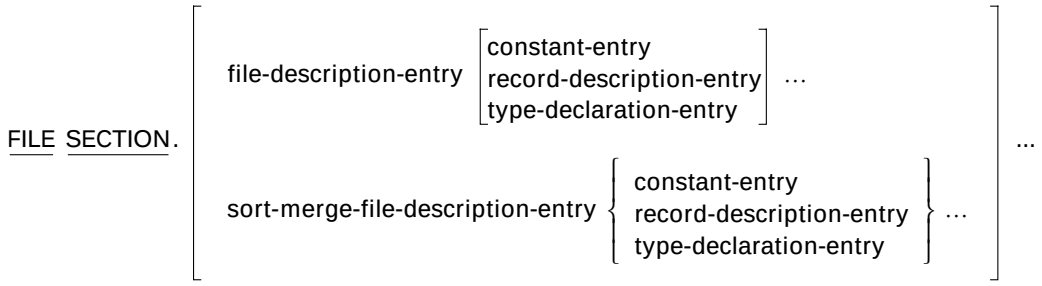
13.3 Explicit and implicit attributes

Attributes may be implicitly or explicitly specified. Any attribute that has been explicitly specified is called an explicit attribute. Some explicit attributes in the description of a group item apply to its subordinate items; these attributes are implicit attributes of the subordinate item. If an attribute has not been explicitly specified for an item or inherited from a group item, the attribute takes on the default specification, which is also known as an implicit attribute. An implicit attribute is treated as if the attribute had been explicitly specified.

13.4 File section

The purpose of the file section is to describe the structure of data, sort, and merge files.

13.4.1 General format



where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
constant-entry	13.10, Constant entry
record-description-entry	13.11, Record description entry
type-declaration-entry	13.12, Type declaration entry

13.4.2 Syntax rules

1) The file section may be specified in a function definition or a program definition. Within a class definition, the file section may be specified only in a factory definition or an instance definition, but not in a method definition. The file section shall not be specified within an interface definition.

13.4.3 General Rules

1) A data-item format or table format VALUE clause specified in the file section is ignored except in the execution of the INITIALIZE statement. The initial value of a data item in the file section is undefined. The initial length of an any-length elementary item is zero.

13.4.4 File description entry

The file description entry (FD entry) represents the highest level of organization in the file section.

The file description entry furnishes information concerning the physical structure, identification, and the internal or external attributes of a file connector, of the associated records, and of the associated data items. The file description entry also determines whether a file-name is a local name or a global name. In addition, for a report file the file description entry furnishes information concerning the physical structure, identification, and report-names pertaining to a report file.

13.4.4.1 General format

Format 1 (sequential):

```
FD file-name-1
  [ IS EXTERNAL [ AS literal-1 ] ]
  [ IS GLOBAL ]
  [ FORMAT { BIT | CHARACTER | NUMERIC } DATA ]
  [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { CHARACTERS | RECORDS } ]
  [ record-clause ]
  [ LINAGE IS { data-name-2 | integer-8 } LINES [ WITH FOOTING AT { data-name-3 | integer-9 } ]
        [ LINES AT TOP { data-name-4 | integer-10 } ] [ LINES AT BOTTOM { data-name-5 | integer-11 } ] ]
  [ CODE-SET { IS alphabet-name-1 [ alphabet-name-2]
             | { FOR ALPHANUMERIC IS alphabet-name-1 | FOR NATIONAL IS alphabet-name-2 } } ] .
```

Format 2 (relative-or-indexed):

```
FD file-name-1
  [ IS EXTERNAL [ AS literal-1 ] ]
  [ IS GLOBAL ]
  [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { CHARACTERS } ]
                                              { RECORDS    }
  [ record-clause ] .
```

Format 3 (report):

```
FD file-name-1
  [ IS EXTERNAL [ AS literal-1 ] ]
  [ IS GLOBAL ]
  [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { CHARACTERS } ]
                                              { RECORDS    }
  [ record-clause ]
  [ CODE-SET { IS alphabet-name-1 [ alphabet-name-2]                                  } ]
             { [ FOR ALPHANUMERIC IS alphabet-name-1 ]                               }
             { [ FOR NATIONAL IS alphabet-name-2     ]                               }
  { REPORT IS  }
  { REPORTS ARE } { report-name-1 } ... .
```

where record-clause is described in 13.18.42, RECORD clause

13.4.4.2 Syntax rules

ALL FORMATS

1) File-name-1 shall be specified in a file control entry.

2) The clauses that follow file-name-1 may appear in any order.

FORMATS 1 AND 2

3) When no record description entries are specified:

   a) a RECORD clause shall be specified in the file description entry,

   b) a FILE phrase specifying file-name-1 and the FROM phrase shall be specified on all WRITE and REWRITE statements associated with the file, and

   c) an INTO phrase shall be specified on all READ statements associated with the file.

FORMAT 1

4) Format 1 is the file description entry for a sequential file.

5) If the FORMAT clause is specified, variable-length records shall be specified for the file.

FORMAT 2

6) Format 2 is the file description entry for a relative file or an indexed file. For an indexed file, one or more record description entries shall be associated with the file description entry.

FORMAT 3

7) Format 3 is the file description entry for a report file. No record description entries or constant entries shall be associated with the file description entry for a report file.

8) The subject of a file description entry that specifies a REPORT clause may be referenced in the procedure division only by the USE statement, the CLOSE statement, or the OPEN statement with the OUTPUT or EXTEND phrase.

13.4.4.3 General rules

ALL FORMATS

1) A file description entry associates file-name-1 with a file connector.

2) If the EXTERNAL clause is specified, all file description entries in the run unit that reference the same file connector as file-name shall obey the following rules:

   a) If any of the file description entries has a BLOCK CONTAINS clause, all shall have a BLOCK CONTAINS clause specifying the same minimum and maximum size of the physical record.

   b) If any of the file description entries has a CODE-SET clause, all shall have a CODE-SET clause specifying the same character set.

   c) If any of the file description entries has a LINAGE clause, all shall have a LINAGE clause specifying

      1. the same corresponding values for any literals specified

      2. the same corresponding external data items.

   d) All file description entries shall have the same smallest and largest record size.

   e) If any of the file description entries has a REPORT clause, all shall have a REPORT clause.

FORMAT 1

3) If the file description entry for a sequential file contains the LINAGE clause and the EXTERNAL clause, the LINAGE-COUNTER data item is an external data item. If the file description entry for a sequential file contains the LINAGE clause and the GLOBAL clause, LINAGE-COUNTER is a global name.

FORMAT 3

4) The report writer logical record structure of the file associated with file-name-1 is defined by the implementor.

13.4.5 Sort-merge file description entry

The sort-merge file description entry (SD entry) represents the highest level of organization in the file section. The sort-merge file description entry furnishes information concerning the physical structure pertaining to a sort or merge file. The clauses of a sort-merge file description entry (SD entry) specify the size and the names of the records associated with a sort file or a merge file. The set of records in the file may reside on external media or in the internal storage of the processor. Storage allocation and record management associated with the file are under control of the SORT/MERGE implementation.

13.4.5.1 General format

```
SD file-name-1
[ RECORD { CONTAINS integer-1 CHARACTERS
         { IS VARYING IN SIZE [ [ FROM integer-2 ] [ TO integer-3 ] CHARACTERS ]
         {     [ DEPENDING ON data-name-1 ]
         { CONTAINS integer-4 TO integer-5 CHARACTERS                            } ] .
```

13.4.5.2 Syntax rules

1) File-name-1 shall be specified in a file control entry.

2) One or more record description entries shall be associated with the sort-merge file description entry.

3) File-name-1 shall not be specified in an input-output statement.

4) A record description entry associated with file-name-1 shall not be specified in an input-output statement other than following the word FROM or the word INTO.

13.4.5.3 General rules

1) The number of characters is specified in terms of bytes.

13.5 Working-storage section

The working-storage section describes records and subordinate data items that are not part of files. Data described in the working-storage section is static or initial data.

13.5.1 General format

```
WORKING-STORAGE SECTION. [ 77-level-description-entry
                           constant-entry
                           record-description-entry
                           type-declaration-entry ] ...
```

where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
77-level-entry	13.13, 77-level data description entry
constant-entry	13.10, Constant entry
record-description-entry	13.11, Record description entry
type-declaration-entry	13.12, Type declaration entry

13.5.2 Syntax rules

1) The working-storage section may be specified in a function definition or a program definition. Within a class definition, the working-storage section may be specified only in a factory definition or an instance definition, but not in a method definition. The working-storage section shall not be specified within an interface definition.

13.5.3 General rules

1) Data items in the working-storage section of a program that does not have the initial attribute, a function, a factory, or an object are static data.

2) Data items in the working-storage section of a program that does have the initial attribute are initial data.

3) Data items in the working-storage section are initialized as indicated in 13.18.62, VALUE clause.

13.6 Local-storage section

The local-storage section describes automatic data.

13.6.1 General format

```
LOCAL-STORAGE SECTION. [ 77-level-description-entry
                         constant-entry
                         record-description-entry
                         type-declaration-entry ] ...
```

where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
77-level-entry	13.13, 77-level data description entry
constant-entry	13.10, Constant entry
record-description-entry	13.11, Record description entry
type-declaration-entry	13.12, Type declaration entry

13.6.2 Syntax rules

1) The local-storage section may be specified in a program definition or function definition or in a method definition contained in a class definition.

13.6.3 General rules

1) Data items in the local-storage section are automatic data.

2) Data items in the local-storage section are initialized as indicated in 13.18.62, VALUE clause.

13.7 Linkage section

The linkage section describes formal parameters and returning items.

Formal parameters and returning items described in the linkage section of a source element are referred to both by that source element, when it is activated, and by the activating source element.

13.7.1 General format

```
LINKAGE SECTION. [ 77-level-description-entry
                   constant-entry
                   record-description-entry
                   type-declaration-entry ] ...
```

where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
77-level-entry	13.13, 77-level data description entry
constant-entry	13.10, Constant entry
record-description-entry	13.11, Record description entry
type-declaration-entry	13.12, Type declaration entry

13.7.2 Syntax rules

1) The linkage section may be specified in a program definition, function definition, method definition, program prototype definition, or function prototype definition.

2) The description of the formal parameters and the returning item that appear in the linkage section of a function prototype or a program prototype shall match the description of the formal parameters and the returning item in the corresponding function definition or program definition, respectively.

3) The description of the parameters and the returning item that appear in a linkage section shall follow the rules specified in 14.8, Conformance for parameters and returning items.

4) A based data item may be referenced as described in 13.18.5, BASED clause; otherwise, a data item defined in the linkage section of a source element may be referenced within the procedure division of that source element if, and only if, it satisfies one of the following conditions:

   a) It is an operand of the USING phrase or the RETURNING phrase of the procedure division header.

   b) It is subordinate to an operand of the USING phrase or the RETURNING phrase of the procedure division header.

   c) It is defined with a REDEFINES or RENAMES clause, the object of which satisfies one of the above conditions.

   d) It is subordinate to any item that satisfies the condition in subrule c.

   e) It is a condition-name or index-name associated with a data item that satisfies one of the above conditions.

5) A formal parameter of a function shall not be used as a receiving operand.

13.7.3 General rules

1) The access to a based data item is described in 13.18.5, BASED clause.

2) The mechanism by which a correspondence is established between the formal parameters and returning items described in the linkage section and data items described in the activating element is described in 14.2.3, General rules of the procedure division. In the case of index-names, no such correspondence is established and index-names in the activated and activating source elements always refer to separate indices.

3) In a program definition, access to formal parameters and the returning item is guaranteed if and only if the program is to execute under the control of a CALL statement, and the CALL statement contains a USING phrase. If a formal parameter or a returning item is accessed in a program that is not a called program, such as a program that is activated by the operating system, the effect is undefined. If a program is activated by a non-COBOL runtime element, the implementor defines whether access to formal parameters and the returning item is guaranteed.

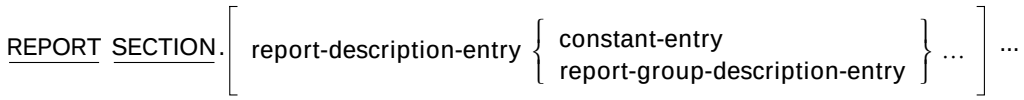
4) In a function definition and in a method definition, access to formal parameters and the returning item is always guaranteed.

5) A data-item format or table format VALUE clause specified in the linkage section is ignored except in the execution of an explicit or implicit INITIALIZE statement. If the runtime element containing the linkage section is activated by a COBOL runtime element, the initial value of a data item in the linkage section is determined by the value of the corresponding formal parameter in the activating runtime element, as described in 14.2.3, General rules of the procedure division. If the runtime element containing the linkage section is activated by the operating system, the initial value of a linkage section data item is undefined. If the runtime element containing the linkage section is activated by a non-COBOL runtime element, the initial value of a linkage section data item is defined by the implementor.

13.8 Report section

The report section describes the reports to be written to report files. The description of each report begins with a report description (RD) entry and is followed by one or more report group descriptions.

13.8.1 General format



where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
77-level-entry	13.13, 77-level data description entry
report-group-description-entry	13.8.4, Report group description entry

13.8.2 Syntax rules

1) The report section may be specified in a function definition or a program definition. Within a class definition, the report section may be specified only in a factory definition or an instance definition, but not in a method definition. The report section shall not be specified within an interface definition.

13.8.3 Report description entry

The report description (RD) entry gives a name to the report and defines its physical and logical subdivisions. (See 13.8.5.1, Physical subdivisions of a report.)

An RD entry shall be followed by one or more report group description entries. The RD entry and the report group description entries that follow fully describe one report. A report is in the active state after the successful execution of an INITIATE statement referencing that report description entry and before the successful execution of a TERMINATE statement referencing that report description entry. At any other time, it is in the inactive state.

13.8.4 Report group description entry

A report group is a block of zero, one, or more report lines that is treated, both logically and visually, as a single unit. Each report group description shall consist of a level 1 report description entry followed by zero, one, or more subordinate entries, describing the vertical and horizontal layout of the report group and the content or origin of each of its printed data items, known as printable items.

The report groups associated with the report are specified immediately following the report description entry. If several report groups are specified, the order in which they are defined is not significant. The first entry of each report group description has level number 1 and a TYPE clause and, if a data-name is also specified, this may be used subsequently to identify the report group. Further subordinate group and elementary entries may be specified to describe additional elements of the report group.

13.8.5 Report subdivisions

A report has physical and logical subdivisions that interact to determine what is printed on a page.

13.8.5.1 Physical subdivisions of a report

13.8.5.1.1 Pages

Each report is composed of pages of equal size, defined by the PAGE clause, or one page of indefinite size. Each page heading, body group (detail, control heading, or control footing), and page footing appears in a separate

subdivision of the page. Each report heading or report footing may appear in any position on the page. Advancing to a new page is executed automatically for body groups, including the printing of a page footing and page heading.

13.8.5.1.2 Lines

Each report group is divided vertically into zero, one, or more lines. Each line, or multiple line set, is represented in the report group description by a LINE clause. The NEXT GROUP clause, where defined, specifies additional vertical spacing following the report group.

13.8.5.1.3 Report Items

Each line of each report group is divided horizontally into zero, one, or more printable items. Each printable item, or adjacent set of printable items, is defined in the report group description by an elementary entry containing a COLUMN clause. The value that is placed in a printable item is determined solely by a SOURCE, SUM, or VALUE clause in its data description entry. In addition, unprintable items may be specified whose entries contain no COLUMN or LINE clause but from which printable items may be derived.

Report items shall not be accessed or referred to by any clauses in any other section of the data division or by any other procedure division statements, except in the following cases:

1) Sum counters may be inspected or altered in the procedure division.

2) Report groups of type DETAIL are accessed by the GENERATE statement.

A report item referenced in a function identifier or inline method invocation shall be an elementary report item.

13.8.5.2 Logical Subdivisions of a Report

Report groups of type DETAIL may be structured into a nested set of control groups. Each control group may begin with a control heading and end with a control footing.

A control break occurs when a change of value is detected in a control data item during the execution of a GENERATE statement. The hierarchy of control data items is used to check automatically for any such change in value. The detection of a control break causes the same GENERATE statement to print each defined control footing, in reverse hierarchical order, and each defined control heading, in hierarchical order.

13.9 Screen section

The screen section describes the screens to be displayed during terminal I-O. The screen section describes screen records and subordinate screen items.

13.9.1 General format

```
SCREEN SECTION. [ constant-entry          ] ...
                [ screen-description-entry ]
```

where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
constant-entry	13.10, Constant entry
screen-description-entry	13.17, Screen description entry

13.9.2 Syntax rules

1) The screen section may be specified in a function definition or a program definition. Within a class definition, the screen section may be specified only in a factory definition or an instance definition, but not in a method definition. The screen section shall not be specified within an interface definition.

13.10 Constant entry

A constant entry defines a constant. A constant may be used in place of a literal.

13.10.1 General format

```
{ 1  }
{ 01 } constant-name-1 CONSTANT [ IS GLOBAL] { AS { arithmetic-expression-1                  }
                                               { BYTE-LENGTH OF data-name-1 }
                                               { literal-1                  }
                                               { LENGTH OF data-name-2      } }
                                               { FROM compilation-variable-name-1 } .
```

13.10.2 Syntax rules

1) If the operand of the constant entry consists of a single numeric literal, that operand is treated as a literal, not as an arithmetic-expression.

2) Except in a compiler directive, constant-name-1 may be used anywhere that a format specifies a literal of the class and category of constant-name-1. If constant-name-1 is an integer, it may also be used to specify repetition in a picture character-string, as specified in 13.18.39, PICTURE clause.

3) Data-name-1 and data-name-2 may be qualified and subscripted. All subscripts of data-name-1 and data-name-2 shall be literals.

4) The length of data-name-1 or data-name-2 shall not be dependent, directly or indirectly, upon the value of constant-name-1.

5) Neither the value of literal-1 nor the value of any of the literals in arithmetic-expression-1 shall be dependent, directly or indirectly, upon the value of constant-name-1.

6) Neither literal-1 nor any of the literals in arithmetic-expression-1 shall be a figurative constant.

7) Arithmetic-expression-1 shall be formed in accordance with 7.3.5, Compile-time arithmetic expressions, with the exception that all of the operands shall be literals and none shall be compilation-variable-names.

8) Compilation-variable-name-1 shall be a compilation-variable-name for which the defined condition is currently true.

9) If constant-name-1 duplicates another constant-name, the specification of arithmetic-expression-1, literal-1, data-name-1, data-name-2, or compilation-variable-name-1 shall be the same as specified in the other constant-name.

10) Data-name-1 and data-name-2 shall not be described with the ANY LENGTH clause.

11) Data-name-1 and data-name-2, if defined in the report section, shall reference elementary report items.

12) Data-name-1 and data-name-2 shall not be any-length elementary items or variable-length groups.

13.10.3 General rules

1) If literal-1 or compilation-variable-name-1 is specified, the effect of specifying constant-name-1 in other than this entry is as if literal-1 or the text represented by compilation-variable-name-1 were written where constant-name-1 is written.

2) If literal-1 or compilation-variable-name-1 is specified, the class and category of constant-name-1 is the same as that of literal-1 or the literal represented by compilation-variable-name-1.

3) If arithmetic-expression-1, data-name-1, or data-name-2 is specified, the effect of writing constant-name-1 in other than this entry is as if an integer literal were written where constant-name-1 is written. This integer literal has the value specified in these general rules.

4) If arithmetic-expression-1 is specified, it is evaluated in accordance with 7.3.5, Compile-time arithmetic expressions, to determine the value of constant-name-1. The class and category of constant-name-1 is numeric. Constant-name-1 is an integer.

5) If the BYTE-LENGTH phrase is specified, the class and category of constant-name-1 is numeric. Constant-name-1 is an integer. The value of constant-name-1 is determined as specified in the BYTE-LENGTH intrinsic function with the exception that when data-name-1 is an occurs-depending group item, the maximum size of the data item is used.

6) If the LENGTH phrase is specified, the class and category of constant-name-1 is numeric. Constant-name-1 is an integer. The value of constant-name-1 is determined as specified in the LENGTH intrinsic function with the exception that when data-name-2 is an occurs-depending group item, the maximum size of the data item is used.

13.11 Record description entry

A record description entry consists of a set of data description entries, the first of which shall have level-number 1, that describe the characteristics of a particular record. Any data item that has been described with level-number 1 and whose declaration does not include the TYPEDEF clause is a record.

A record description may have a hierarchical structure. The structure of a record description and the elements allowed in a record description entry are explained in 8.5.1.2, Levels, and in 13.16, Data description entry.

Data elements that bear no hierarchical relationship to any other data item may be described as records that are single elementary items. Alternatively, such data elements when defined in the working-storage section, local-storage section, and linkage section may be described as separate data description entries having level-number 77, as described in 13.13, 77-level data description entry.

13.12 Type declaration entry

A type declaration entry is a data description entry that contains a TYPEDEF clause. A type declaration entry may have a hierarchical structure beginning with level-number 1. The structure of a type declaration entry and the elements allowed in a type declaration entry are explained in 8.5.1.2, Levels, and in 13.16, Data description entry.

A type declaration entry has no storage associated with it.

Further details regarding the specification of a type declaration entry are described in 13.18.57, TYPEDEF clause.

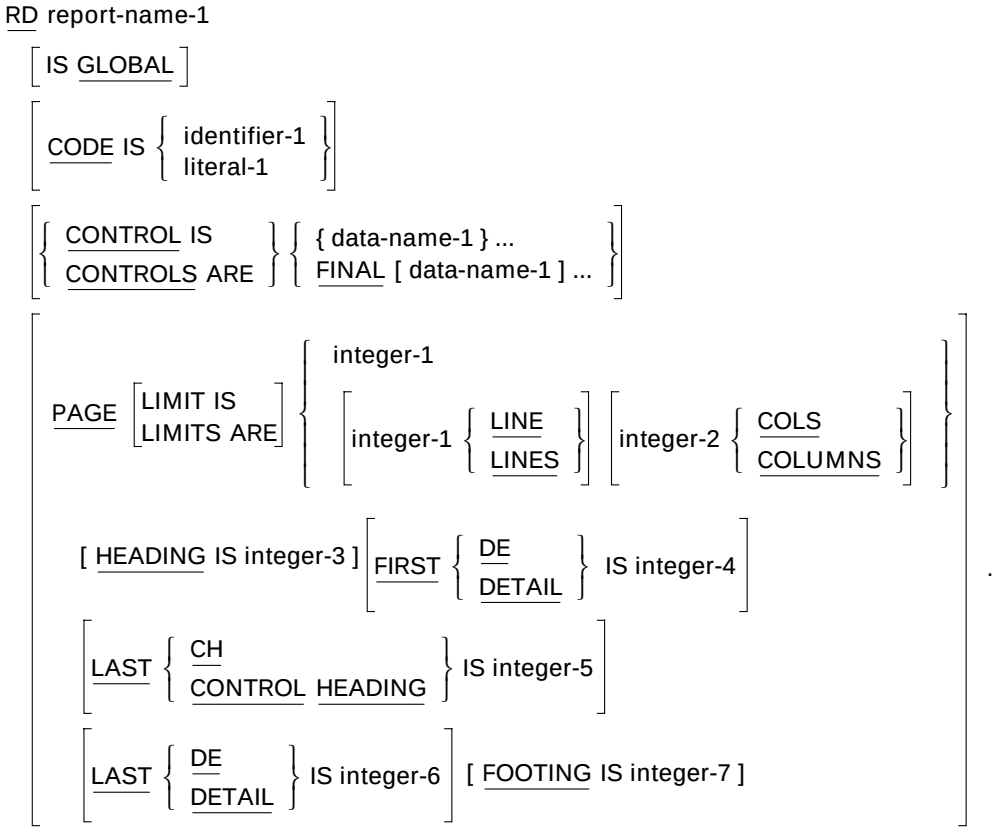
13.13 77-level data description entry

Items in the linkage section, local-storage, and the working-storage section that bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary data items. Each of these items is defined in a separate data description entry that begins with the special level-number 77.

13.14 Report description entry

The report description entry names a report and describes its general physical and logical structure.

13.14.1 General format



13.14.2 Syntax rules

1) There shall be one and only one REPORT clause specifying report-name-1 in a given file description entry.

2) The clauses that follow report-name-1 may appear in any order.

13.14.3 General rules

1) If GLOBAL is specified, report-name-1 and all its constituent report groups, its PAGE-COUNTER and LINE-COUNTER, and any sum counters defined in report-name-1 are global.

13.15 Report group description entry

The report group description entry specifies the characteristics of a report group and of the individual items within a report group.

13.15.1 General format

```
level-number [ entry-name-clause ]
  [type-clause]
  [next-group-clause]
  [line-clause]
  [picture-clause]
  [ [ USAGE IS ] { DISPLAY  }  ]
                 { NATIONAL }
  [sign-clause]
  [justified-clause]
  [column-clause]
  [BLANK WHEN ZERO]
  [source-clause  ]
  [sum-clause     ]
  [value-clause   ]
  [PRESENT WHEN condition-1]
  [GROUP INDICATE]
  [OCCURS [ integer-1 TO ] integer-2 TIMES [ DEPENDING ON data-name-1 ] [ STEP integer-3 ]]
  [varying-clause] .
```

where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
column-clause	13.18.14, COLUMN clause
entry-name-clause	13.18.19, Entry-name clause
justified-clause	13.18.31, JUSTIFIED clause
line-clause	13.18.34, LINE clause
next-group-clause	13.18.36, NEXT GROUP clause
picture-clause	13.18.39, PICTURE clause
sign-clause	13.18.51, SIGN clause
source-clause	13.18.52, SOURCE clause
sum-clause	13.18.53, SUM clause
type-clause	13.18.56, TYPE clause (report-group format)
value-clause	13.18.62, VALUE clause (report-section format)
varying-clause	13.18.63, VARYING clause

13.15.2 Syntax rules

1) Report group description entries may appear only in the report section.

2) If the entry-name clause is specified, the data-name format or filler format shall be specified. The entry-name clause shall immediately follow the level-number. All other clauses may be written in any order.

3) Level-number shall be any integer from 1 through 49.

4) The first entry that follows a report description entry shall be a level 1 entry.

5) The TYPE clause may be specified only in a level 1 entry and shall be specified in every level 1 entry.

6) The NEXT GROUP clause may be specified only in a level 1 entry.

7) The data-name format of the entry-name clause shall be specified when the data-name is referenced in a GENERATE statement, a USE BEFORE REPORTING statement, as a qualifier for a SUM counter, in the UPON phrase of the SUM clause, or as an operand in a SUM clause. The data-name shall not be referenced in any other way.

8) No report group description entry with a LINE clause shall be subordinate to another entry with a LINE clause.

9) Every elementary entry with a COLUMN clause but no LINE clause shall be subordinate to an entry with a LINE clause.

10) Every elementary entry with a COLUMN clause shall also contain either a SOURCE, VALUE or SUM clause.

11) The PICTURE, COLUMN, SOURCE, VALUE, SUM, and GROUP INDICATE clauses may be written only in an elementary entry.

12) A PICTURE clause shall be specified in every elementary entry that has a SOURCE or SUM clause.

13) A COLUMN clause shall be specified in each elementary entry that has a VALUE clause.

14) The PICTURE clause may be omitted for an elementary item when an alphanumeric, boolean or national literal is specified in the VALUE clause. A PICTURE clause is implied as follows:

    a) If the literal is alphanumeric, 'PICTURE X(length)'

    b) If the literal is boolean, 'PICTURE 1(length)'

    c) If the literal is national, 'PICTURE N(length)'

    where length is the length of the literal as specified in 8.3.1.2, Literals.

15) If BLANK WHEN ZERO or JUSTIFIED is specified, a COLUMN clause shall also be specified.

16) Condition-1 shall not reference any sum counter, LINE-COUNTER, PAGE-COUNTER, or other report section data item.

17) The GROUP INDICATE clause shall not be specified in an entry in which the PRESENT WHEN clause is specified.

13.15.3 General rules

1) Each level 1 entry identifies a report group. The report group is defined by this entry and all its subordinate entries.

2) Except for the additional restrictions defined under syntax rules above, the USAGE, PICTURE, BLANK WHEN ZERO and JUSTIFIED clauses are the same clauses as those that are described under the general format for a data description entry and shall obey the syntax rules and general rules defined for each clause. (See 13.18, Data division clauses.)

3) An entry that contains either an OCCURS clause or a LINE or COLUMN clause with more than one operand is said to be a repeating entry, and the number of repetitions is defined to be integer-2 of the OCCURS clause or the number of operands of the LINE or COLUMN clause, whichever is applicable. The number of repetitions of an entry that is not a repeating entry is defined to be 1. A report item is a repeating item if it is defined by a repeating entry or by an entry that is subordinate to a repeating entry.

13.16 Data description entry

A data description entry specifies the characteristics of a particular item of data. A level 1 data description entry within the file, working-storage, local-storage, or linkage section determines whether the record and its subordinate data items have local names or global names.

A level 1 data description entry in the working-storage section determines the internal or external attribute of the record and its subordinate data items.

13.16.1 General format

Format 1 (data-description):

```
level-number [ entry-name-clause]
  [ REDEFINES data-name-1 ]
  [ IS TYPEDEF [ STRONG ] ]
  [ ALIGNED ]
  [ ANY LENGTH ]
  [ BASED ]
  [ BLANK WHEN ZERO ]
  [ CONSTANT RECORD ]
  [ IS EXTERNAL [ AS literal-1 ] ]
  [ IS GLOBAL ]
  [ GROUP-USAGE IS { BIT      } ]
                   { NATIONAL }
  [ { JUSTIFIED } RIGHT ]
    { JUST      }
  [ occurs-clause ]
  [ picture-clause ]
  [ PROPERTY [ WITH NO { GET } ] [ IS FINAL ] ]
                       { SET }
  [ SAME AS data-name-3 ]
  [ select-when-clause ]
  [ [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
                { TRAILING }
  [ { SYNCHRONIZED } [ LEFT  ] ]
    { SYNC         } [ RIGHT ]
  [ TYPE type-name-1 ]
  [ usage-clause ]
  [ validation-clauses ]
  [ value-clause ] .
```

where validation-clauses is:

[class-clause]
[default-clause]
[DESTINATION IS { identifier-2 } ...]
[{ INVALID WHEN condition-2 } ...]
[PRESENT WHEN condition-3]
[VARYING { data-name-6 [FROM arithmetic-expression-1] [BY arithmetic-expression-2] } ...]
[validate-status-clause] ...

where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
class-clause	13.18.11, CLASS clause
default-clause	13.18.17, DEFAULT clause
entry-name-clause	13.18.19, Entry-name clause
occurs-clause	13.18.37, OCCURS clause (fixed-table, occurs-depending-table, or dynamic-capacity-table-format)
picture-clause	13.18.39, PICTURE clause
select-when clause	13.18.50, SELECT WHEN clause
usage-clause	13.18.59, USAGE clause
validate-status-clause	13.18.61, VALIDATE-STATUS clause
value-clause	13.18.62, VALUE clause (data-item or table format)

Format 2 (renames):

66 data-name-1 RENAMES data-name-4 [{ THROUGH / THRU } data-name-5] .

Format 3 (condition-name):

88 condition-name-1 value-clause .

where value-clause is described in 13.18.62, VALUE clause (condition-name format).

Format 4 (validation):

88 [condition-name-2] value-clause .

where value-clause is described in 13.18.62, VALUE clause (content-validation-entry format).

13.16.2 Syntax rules

FORMAT 1

1) Level-number may be 77 or 1 through 49.

2) The data-name format of the entry-name clause shall be specified if level-number is 77.

3) The REDEFINES clause shall not be specified in the same data description entry as the BASED, CONSTANT RECORD, or TYPEDEF clause.

4) If the entry-name clause is specified, either the data-name format or the filler format may be specified and the entry-name clause shall immediately follow the level-number. If no entry-name clause is specified, it is as though the filler format of the entry-name clause were specified. The REDEFINES clause, if specified, shall immediately follow the entry-name clause, if specified; otherwise, the REDEFINES clause shall immediately follow the level-number. If the TYPEDEF clause is specified, the data-name format of the entry-name clause shall also be specified and the TYPEDEF clause shall immediately follow the entry-name clause. The remaining clauses may be written in any order.

5) The EXTERNAL clause shall not be specified in the same data description entry as the REDEFINES, TYPEDEF, or BASED clause.

6) The CONSTANT RECORD and GLOBAL clauses may be specified only in data description entries whose level-number is 1.

7) The data-name format of the entry-name clause shall be specified for any entry containing the GLOBAL or EXTERNAL clause, or for record descriptions associated with a file description entry that contains the EXTERNAL or GLOBAL clause.

8) The PICTURE clause shall not be specified for the subject of a RENAMES clause or for an item whose usage is binary-char, binary-short, binary-long, binary-double, float-short, float-long, float-extended, index, object reference, pointer, function-pointer, or program-pointer. For any other entry describing an elementary item, a PICTURE clause shall be specified except as indicated in syntax rule 9.

9) The PICTURE clause may be omitted for an elementary item when an alphanumeric, boolean, or national literal is specified in the data-item format of the VALUE clause. A PICTURE clause is implied as follows:

   a) if the literal is alphanumeric, 'PICTURE X(length)'

   b) if the literal is boolean, 'PICTURE 1(length)'

   c) if the literal is national, 'PICTURE N(length)'

   where length is the length of the literal as specified in 8.3.1.2, Literals.

10) The VALUE clause shall not be specified for data items of class index, object, or pointer.

11) The SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses may be specified only for an elementary data item.

12) The SAME AS clause shall not be specified in the same data description entry with any clauses except CONSTANT RECORD, entry-name, EXTERNAL, GLOBAL, level-number, and OCCURS.

13) The ANY LENGTH, BASED, BLANK WHEN ZERO, select-when, SYNCHRONIZED, and TYPEDEF clauses and validation-clauses shall not be specified in the same data description entry with the CONSTANT RECORD clause, or in any data description entry subordinate to a data description entry with the CONSTANT RECORD clause.

14) The TYPE clause shall not be specified in the same data description entry with any clauses except BASED, CLASS, CONSTANT RECORD, DEFAULT, DESTINATION, entry-name, EXTERNAL, GLOBAL, INVALID, level-number, OCCURS, PRESENT WHEN, PROPERTY, TYPEDEF, VALIDATE-STATUS, VALUE, and VARYING.

15) The TYPEDEF clause may be specified only in a data description entry whose level-number is 1 and for which the data-name format of the entry-name clause is specified.

16) The BASED clause may be specified only in data description entries in the linkage section, in the working-storage section, and in the local-storage section. The level number of such data description entries shall be 1 or 77.

17) If the ANY LENGTH clause is specified, the only other clauses permitted are level-number, entry-name, PICTURE, USAGE, and VALUE.

18) If the LOCALE phrase of the PICTURE clause is specified, the SIGN clause shall not be specified.

19) The PRESENT WHEN clause shall not be specified in a data description entry that has a level number of 1 or 77.

20) The PROPERTY clause shall not be specified in the same data description entry as:

    a) a BASED clause,

    b) a TYPEDEF clause.

FORMAT 2

21) The words THROUGH and THRU are equivalent.

FORMATS 3 AND 4

22) Each condition-name is subordinate to the data-name with which it is associated.

23) Format 3 or 4 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. The condition-name entries for a particular conditional variable shall immediately follow the entry describing the item with which the condition-name is associated. A condition-name may be associated with any data description entry that contains a level-number except the following:

    a) Another level 88 entry.

    b) A level 66 entry.

    c) An alphanumeric group containing items with a usage other than display.

    d) A group containing items described with a JUSTIFIED or SYNCHRONIZED clause.

    e) A data item of the class index, object, or pointer.

    f) A data item described with the ANY LENGTH clause.

    g) A type declaration described with the STRONG phrase, or a group item subordinate to such a type declaration.

    h) A variable-length group.

13.16.3 General rules

1) If the subject of the entry is a group item that is subordinate to a bit group and a GROUP-USAGE BIT clause is not specified, a GROUP-USAGE BIT clause is implied for the subject of the entry.

2) If the subject of the entry is a group item that is subordinate to a national group and a GROUP-USAGE NATIONAL clause is not specified, a GROUP-USAGE NATIONAL clause is implied for the subject of the entry.

3) Format 3 contains the name of the condition and the value, values, or range of values associated with the condition-name.

4) Format 4 may contain the name of a condition and the value, values, or range of values associated with the condition-name, in which case the condition-name has the same meaning as in Format 3 and may be used in the same way. During the content validation stage of a VALIDATE statement that references the subject of the

entry or a superordinate data item, the value or values specified in this format define the value, values, or range of values that make the subject of the entry valid if VALID is specified or invalid if INVALID is specified.

13.17 Screen description entry

A screen description entry specifies attributes, behavior, size, and location of a screen item so that it can be referenced by an ACCEPT screen or a DISPLAY screen statement. The screen description entry allows data items to be associated with the screen item so that the contents of the data item are displayed within the screen item or the value keyed into a screen item by the operator is placed in the data item.

13.17.1 General format

Format 1 (group):

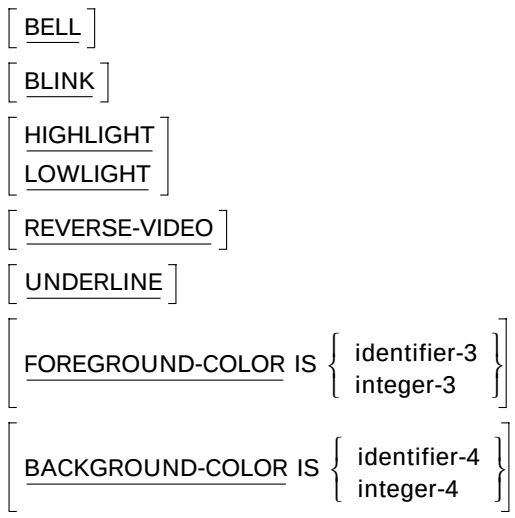
```
level-number [ entry-name-clause ]
  [ IS GLOBAL ]
  [ LINE NUMBER IS [ PLUS | + | MINUS | – ] { identifier-1 | integer-1 } ]
  [ { COLUMN | COL } NUMBER IS [ PLUS | + | MINUS | – ] { identifier-2 | integer-2 } ]
  [ BLANK SCREEN ]
  [ screen-attribute-clauses ]
  [ [ SIGN IS ] { LEADING | TRAILING } [ SEPARATE CHARACTER ] ]
  [ FULL ]
  [ AUTO ]
  [ SECURE ]
  [ REQUIRED ]
  [ OCCURS integer-5 TIMES ]
  [ [ USAGE IS ] { DISPLAY | NATIONAL } ] .
```

Format 2 (elementary):

```
level-number [ entry-name-clause ]
  [ IS GLOBAL ]
  [ LINE NUMBER IS [ PLUS | + | MINUS | - ] { identifier-1 | integer-1 } ]
  [ { COLUMN | COL } NUMBER IS [ PLUS | + | MINUS | - ] { identifier-2 | integer-2 } ]
  [ BLANK { LINE | SCREEN } ]
  [ ERASE { END OF LINE | END OF SCREEN | EOL | EOS } ]
  [ screen-attribute-clauses ]
  [ picture-clause ]
  [ source-destination-clauses ]
  [ BLANK WHEN ZERO ]
  [ { JUST | JUSTIFIED } RIGHT ]
  [ [ SIGN IS ] { LEADING | TRAILING } [ SEPARATE CHARACTER ] ]
  [ FULL ]
  [ AUTO ]
  [ SECURE ]
  [ REQUIRED ]
  [ OCCURS integer-5 TIMES ]
  [ [ USAGE IS ] { DISPLAY | NATIONAL } ] .
```

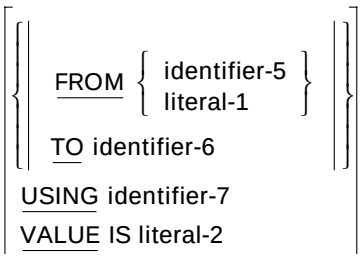
where entry-name-clause is described in 13.18.19, Entry-name clause

where screen-attribute-clauses is:



where picture-clause is described in 13.18.39, PICTURE clause.

where source-destination-clauses is:



13.17.2 Syntax rules

ALL FORMATS

1) The entry-name clause, if specified, shall immediately follow level-number. If the entry-name clause is specified, only the screen-name format or filler format may be specified. If the entry-name clause is not specified, it is as if the FILLER format of the entry-name clause were specified. The remaining clauses may be specified in any order.

2) If the GLOBAL clause is specified, the screen-name format of the entry-name clause shall be specified and level-number shall be 1.

FORMAT 1

3) Level-number shall be a number from 1 through 48.

4) The subject of the entry shall be a group screen item.

FORMAT 2

5) Level-number shall be a number from 1 through 49.

6) The subject of the entry shall be an elementary screen item.

7) For an elementary screen item, the associated screen description entry shall include at least one of the following:

   — a PICTURE clause and a FROM, TO, or USING clause;
   — a PICTURE clause and a VALUE clause with a numeric literal;
   — a VALUE clause specifying an alphanumeric, boolean, or national literal;
   — a BLANK clause;
   — a ERASE clause;
   — a BELL clause.

8) If the FULL clause is specified, the JUSTIFIED clause shall not be specified.

9) If the LOCALE phrase of the PICTURE clause is specified, the SIGN clause shall not be specified.

13.17.3 General rules

ALL FORMATS

1) If the same clause, other than an OCCURS clause, is specified at more than one level in the hierarchy of a screen item, the clause that appears at the lowest level of the hierarchy is the one that takes effect.

2) If the HIGHLIGHT and LOWLIGHT clauses are both specified in the hierarchy of a screen item, the clause that appears at the lowest level of the hierarchy is the one that takes effect.

FORMAT 2

3) The PICTURE clause may be omitted when an alphanumeric, boolean, or national literal is specified in the VALUE clause. A PICTURE clause is implied as follows:

   a) if the literal is alphanumeric, 'PICTURE X(length)'

   b) if the literal is boolean, 'PICTURE 1(length)'

   c) if the literal is national, 'PICTURE N(length)'

   where length is the length of the literal as specified in 8.3.1.2, Literals.

13.18 Data division clauses

13.18.1 ALIGNED clause

The ALIGNED clause specifies that a bit group item or an elementary bit data item is to be aligned at the first bit of the first available byte.

13.18.1.1 General format

ALIGNED

13.18.1.2 Syntax rules

1) The ALIGNED clause may be specified only for a bit group item or an elementary bit data item.

13.18.1.3 General rules

1) An ALIGNED clause causes the subject of the entry to be aligned on the first bit of the first available byte boundary. Implicit filler bits may be generated to complete the assignment of bits as described in 8.5.1.5.3, Alignment of data items of usage bit.

2) An ALIGNED clause specified for a multiple-occurrence data item applies to each occurrence of that item.

3) When an ALIGNED clause is not specified, bit data items are aligned in accordance with 8.5.1.5.3, Alignment of data items of usage bit.

13.18.2 ANY LENGTH clause

The ANY LENGTH clause specifies that the length of a data item can vary at runtime.

13.18.2.1 General format

```
ANY LENGTH [ LIMIT IS integer-1 ]
    [ any-length-structure-name-1 ]
```

13.18.2.2 Syntax rules

1) A PICTURE clause shall be specified for the subject of the entry. The character-string specified in that PICTURE clause shall be one instance of the picture symbol 'N', 'X', or '1'.

2) An any-length-structure-name-1 shall be described if the subject of the entry is specified in the file section, or the subject of the entry is described in the linkage section and is referenced in its procedure division header as a formal parameter passed by value.

3) If any-length-structure-name-1 and the LIMIT phrase are both specified, integer-1 shall not be greater than the maximum length associated with any-length-structure-name-1.

4) For data items of class Boolean:

   a) The ANY LENGTH clause may be specified only in an elementary level 1 entry in the linkage section of a function, of a contained program, or of a method that is not a property method.

   b) The ANY LENGTH clause shall be specified without any additional phrases.

   c) If the source element containing the ANY LENGTH clause is a contained program or is a method, the subject of the entry shall be referenced in its procedure division header as either

      1. a formal parameter with the BY REFERENCE phrase, or

      2. the returning item.

   d) If the source element containing the ANY LENGTH clause is a function, the subject of the entry shall be referenced in its procedure division header as a formal parameter with the BY REFERENCE phrase.

13.18.2.3 General rules

1) The ANY LENGTH clause specifies that the length of the data item defined by the entry may vary. The minimum length of the data item is zero. The picture symbol determines the class

2) Integer-1 specifies the maximum number of characters that may be contained by the subject of the entry. If the LIMIT phrase is not specified, the maximum number of characters that may be contained by the subject of the entry is implementor-defined.

3) The INDIRECT phrase specifies that the subject of the entry is an indirect elementary data item as defined in 8.5.1.7.2, Location of any-length elementary items. If the INDIRECT phrase is not specified, an any-length elementary item declared at the 01 or 77 level or containing the LIMIT phrase is a direct elementary data item; otherwise, that data item is an indirect elementary data item.

4) The PREFIXED phrase specifies that the current length of the any-length data item is contained in a binary data item that directly precedes the data area for an any-length data item. If BINARY-CHAR is specified, the usage of this data item is BINARY-CHAR. If BINARY-SHORT is specified or implied, the usage of this data item is BINARY-SHORT. If BINARY-LONG is specified, the usage of this data item is BINARY-LONG.

5) The DELIMITED phrase specifies that a delimiter directly follows the data area for an any-length data item. The default delimiter for alphanumeric any-length elementary items is X'00'. The default delimiter for national any-length elementary items is NX'0000'.

13.18.3 AUTO clause

The AUTO clause causes the cursor to be automatically moved to the next field declared for the screen item during execution of an ACCEPT screen statement.

13.18.3.1 General format

AUTO

13.18.3.2 General rules

1) An AUTO clause specified at the group level applies to each input screen item in that group.

2) The AUTO clause is ignored for a field that is not an input field.

3) The AUTO clause takes effect during the execution of an ACCEPT screen statement that references the screen item for which the AUTO clause is specified.

4) The AUTO clause causes the cursor to be automatically moved to the next input field declared for the screen item when the last character of the input field whose definition contains this clause has data entered into it.

5) When the AUTO clause is specified for an input field that has no logical next field during input, then when that field is available for input during an ACCEPT screen statement and data is entered into the last character of the screen item, successful completion with normal termination of the ACCEPT statement results.

13.18.4 BACKGROUND-COLOR clause

The BACKGROUND-COLOR clause specifies the background color for the screen item.

13.18.4.1 General format

BACKGROUND-COLOR IS { identifier-1 / integer-1 }

13.18.4.2 Syntax rules

1) Identifier-1 shall be described in the file, working-storage, local-storage, or linkage section as an unsigned integer data item.

2) Integer-1 shall have a value in the range of 0 to 7.

13.18.4.3 General rules

1) When an ACCEPT screen statement or a DISPLAY screen statement referencing the associated screen item is executed, the content of the data item referenced by identifier-1 shall be in the range of 0 to 7.

2) Integer-1 or the content of the data item referenced by identifier-1 specifies the color number of the background color to be used for displaying the screen item. The color associated with the color number is specified in 9.2.7, Color number.

3) A BACKGROUND-COLOR clause specified at the group level applies to each elementary screen item in that group.

4) When a BACKGROUND-COLOR clause is not specified or the value is not in the range 0 to 7, the background color is implementor-defined.

13.18.5 BASED clause

The BASED clause defines a based entry. A based entry is a template that, when associated with an actual data item or allocated storage, describes a based data item.

13.18.5.1 General format

BASED

13.18.5.2 Syntax rules

1) The subject of the entry shall not be of class object.

2) The subject of the entry shall not be an any-length elementary item or a variable-length group.

13.18.5.3 General rules

1) The BASED clause specifies that the subject of the entry is a based entry defining a template that is dynamically associated with storage through an implicit data-address pointer.

2) An implicit data-address pointer is created and has an initial value of NULL. The life cycle of this implicit data-address pointer is described in 8.6.4, Based entries and based data items.

3) If the subject of the entry or any item subordinate to it is referenced directly or indirectly while its address is NULL, the EC-DATA-PTR-NULL exception condition is set to exist.

4) If the subject of the entry is referenced while its address is not NULL and not a valid address of storage, the EC-BOUND-PTR exception condition is set to exist.

13.18.6 BELL clause

The BELL clause causes the terminal audio tone to sound.

13.18.6.1 General format

BELL

13.18.6.2 General rules

1) The use of this clause results in the audio tone sounding when the screen item in which it is specified is processed during the execution of a DISPLAY screen statement. The audio tone sounds once at the start of the execution of the DISPLAY screen statement regardless of how many entries specify the clause.

2) A BELL clause specified at the group level applies to each elementary screen item in that group.

13.18.7 BLANK clause

The BLANK clause clears a screen line or clears the whole screen during the execution of a DISPLAY screen statement before data is transferred to the screen item.

13.18.7.1 General format

```
BLANK { LINE   }
      { SCREEN }
```

13.18.7.2 General rules

1) When the BLANK LINE clause is specified, the entire line specified for the screen item that is the subject of the entry, columns 1 through the end of the line, is cleared during the execution of a DISPLAY screen statement before data is transferred to the screen item.

2) When the BLANK SCREEN clause is specified, the screen is cleared and the cursor is placed at line 1, column 1 during the execution of a DISPLAY screen statement before data is transferred to the screen item. Upon clearing the screen, the background color for the entire screen is set to the value applicable at the time.

3) The BLANK SCREEN clause in combination with the BACKGROUND-COLOR clause for the same screen item, or for a screen item to which it is subordinate, establishes the default background color to be used until the same combination is encountered specifying another background color.

4) The BLANK SCREEN clause in combination with the FOREGROUND-COLOR clause for the same screen item, or for a screen item to which it is subordinate, establishes the default foreground color to be used until the same combination is encountered specifying another foreground color.

5) The BLANK clause is ignored during all phases of the execution of an ACCEPT screen statement.

13.18.8 BLANK WHEN ZERO clause

The BLANK WHEN ZERO clause causes the blanking of an item when a value of zero is being stored in it.

13.18.8.1 General format

BLANK WHEN ZERO

13.18.8.2 Syntax rules

1) The BLANK WHEN ZERO clause may be specified only for an elementary item described by its picture character-string as category numeric-edited or as numeric without the picture symbol 'S'.

2) The subject of the entry shall be implicitly or explicitly described as usage display or usage national.

13.18.8.3 General rules

1) When the BLANK WHEN ZERO clause is specified for a data item, the content of the data item is set to all spaces when the item is a receiving operand and the value being stored is zero.

2) If the subject of the entry is described by its picture character-string as category numeric, the BLANK WHEN ZERO clause defines the item as numeric-edited.

13.18.9 BLINK clause

The BLINK clause specifies that each character of the field blinks when it is displayed on the screen.

13.18.9.1 General Format

BLINK

13.18.9.2 General Rules

1) A BLINK clause specified at the group level applies to each elementary screen item in that group.

2) When the BLINK clause is specified, the characters that constitute the screen item will blink when the screen item is referenced in an ACCEPT screen or a DISPLAY screen statement.

13.18.10 BLOCK CONTAINS clause

The BLOCK CONTAINS clause specifies the size of a physical record.

13.18.10.1 General format

```
BLOCK CONTAINS [ integer-1 TO ] integer-2 { CHARACTERS }
                                          { RECORDS    }
```

13.18.10.2 Syntax rules

1) If integer-1 is specified, integer-2 shall be greater than integer-1.

13.18.10.3 General rules

1) This clause is required except when one or more of the following conditions exist:

   a) A physical record contains one and only one complete logical record.

   b) The hardware device assigned to the file has one and only one physical record size.

   c) The number of records or characters contained in a block is specified in the operating environment.

2) The size of a physical record may be stated in terms of records unless one or more of the following situations exists, in which case the RECORDS phrase shall not be used:

   a) In mass storage files, logical records may extend across physical records.

   b) The physical record contains padding (area not contained in a logical record).

   c) Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

3) If the CHARACTERS phrase is specified, the physical record size is specified in terms of the number of bytes required to store the physical record, regardless of the types of characters used to represent the items within the physical record.

4) If integer-1 is not specified, integer-2 represents the exact size of the physical record. If integer-1 and integer-2 are both specified, they refer to the minimum and maximum size of the physical record, respectively.

13.18.11 CLASS clause

The CLASS clause specifies a range of values for each character of a data item, to be checked during the content validation stage of the execution of a VALIDATE statement.

13.18.11.1 General format

```
CLASS IS { NUMERIC
         | ALPHABETIC
         | ALPHABETIC-LOWER
         | ALPHABETIC-UPPER
         | BOOLEAN
         | alphabet-name-1
         | class-name-1 }
```

13.18.11.2 Syntax rules

1) The subject of the entry shall be of class alphabetic, alphanumeric, or national.

13.18.11.3 General rules

1) The CLASS clause takes effect during the content validation stage of the execution of a VALIDATE statement. The CLASS clause is ignored during the execution of any statement other than a VALIDATE statement.

2) If the subject of the entry is an elementary item, the CLASS clause takes effect only if the item's associated internal indicator is still set to its initial valid value, indicating that the item is valid on format.

3) The operand of the CLASS clause is used to check each character of the value of the data item as described in 8.8.4.1.3, Class condition. If the class condition is false, the data item's internal indicator is set to invalid on content.

4) If the subject of the entry is an alphanumeric group item or national group item, general rules 2 and 3 are applied to each of its subordinate elementary items, except for any data items that are not processed as a result of a PRESENT WHEN clause or an OCCURS clause with a DEPENDING phrase.

13.18.12 CODE clause

The CODE clause specifies one or more characters used to separate multiple reports written to the same file or for specifying correct functioning of the printer device.

13.18.12.1 General format

CODE IS { literal-1 / identifier-1 }

13.18.12.2 Syntax rules

1) Literal-1 shall be an alphanumeric literal.

2) Identifier-1 shall reference an alphanumeric data item that shall not be an occurs-depending-on group item or an any-length elementary item.

3) If the CODE clause is specified for any report, it shall be specified for each report associated with the same report file.

13.18.12.3 General rules

1) When the CODE clause is specified, literal-1 or identifier-1 is automatically placed in the first characters of each logical record written to the report file for this report.

2) The characters occupied by literal-1 or identifier-1 are not included in the descriptions of the lines in the report, but are included in the logical record size.

3) If identifier-1 is specified, it is evaluated at the start of the processing for each body group, either during page advance processing, as detailed in the GENERATE statement, or whenever page advance processing is not performed. The resultant value is used until the next evaluation.

13.18.13 CODE-SET clause

The CODE-SET clause specifies the character code convention used to represent data on the external media.

13.18.13.1 General format

```
CODE-SET { IS alphabet-name-1 [ alphabet-name-2 ]
         { [ FOR ALPHANUMERIC IS alphabet-name-1 ]
         { [ FOR NATIONAL IS alphabet-name-2 ]
```

13.18.13.2 Syntax rules

1) Alphabet-name-1 shall reference an alphabet that defines an alphanumeric coded character set.

2) Alphabet-name-2 shall reference an alphabet that defines a national coded character set.

3) If there are record description entries associated with the file and no SELECT WHEN clauses are specified, either alphabet-name-1 or alphabet-name-2 may be specified, but not both; and:

   a) if alphabet-name-1 is specified, all elementary data items of all record description entries associated with the file shall be described as usage display, and any signed numeric data items shall be described with the SIGN IS SEPARATE clause.

   b) if alphabet-name-2 is specified, all elementary data items of all record description entries associated with the file shall be described as usage national, and any signed numeric data items shall be described with the SIGN IS SEPARATE clause.

13.18.13.3 General rules

1) The CODE-SET clause identifies alphabets to be used for converting data from a coded character set on the storage medium to the native character set during input operations, and from the native character set to the coded character set on the storage medium during output operations.

2) Upon successful processing of an OPEN statement for the file referenced in this file description entry, the coded character set used to represent alphanumeric data on the storage medium is the one referenced by alphabet-name-1; the coded character set used to represent national data on the storage medium is the one referenced by alphabet-name-2.

3) If the record description entries associated with the file do not contain a SELECT WHEN clause, the specified alphabet is used for code-set conversion of all data items in each record.

4) If record description entries associated with the file contain a SELECT WHEN clause, a record description entry is selected by evaluation of those SELECT WHEN clauses. If there are no record description entries associated with the file, the record description used for conversion is the description of the identifier or literal specified in the FROM phrase of a WRITE or REWRITE statement specifying the FILE phrase.

   Alphabet-name-1 is used for code-set conversion of each data item described with usage display in the selected record description entry. Alphabet-name-2 is used for code-set conversion of each data item described with usage national in the selected record description entry.

   NOTE There is no conversion of data items described with other usages unless a FORMAT clause applies.

5) If a FORMAT clause is also specified for the file, the logical order of processing for each data item is:

   — formatting first, then code-set conversion during output operations;

   — code-set conversion first, then formatting during input operations.

6) For each data item to be converted:

   a) On input, each coded character from the storage medium is replaced with its associated native coded character as defined in the alphabet being used.

   b) On output, each native coded character in the record is replaced for the storage medium with its associated coded character as defined in the alphabet being used.

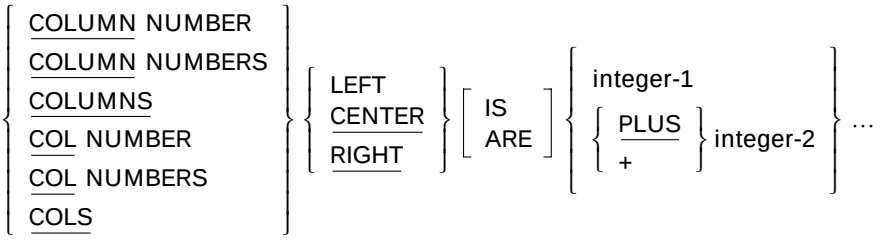
7) If the CODE-SET clause is not specified, the native character set is assumed for data on the external media.

13.18.14 COLUMN clause

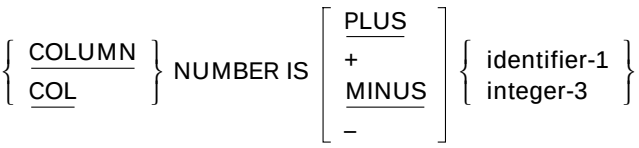
The COLUMN clause identifies a printable item, or a set of printable items, and specifies their horizontal location in a report line, or specifies the horizontal screen coordinate for a screen item.

13.18.14.1 General format

Format 1 (report-writer):



Format 2 (screen-item):



13.18.14.2 Syntax rules

ALL FORMATS

1) COLUMN, COL, COLUMNS, and COLS are synonyms.

2) PLUS and + are synonyms.

FORMAT 1

3) The COLUMN clause may be specified only in an elementary entry. The entry shall also contain a LINE clause or shall be subordinate to an entry containing a LINE clause.

4) The keyword ARE may be specified only if COLUMNS, COLS, or NUMBERS is specified.

5) The keyword IS shall not be specified if COLUMNS, COLS, or NUMBERS is specified.

6) Neither integer-1 nor integer-2 shall exceed the page width. (See 13.18.38, PAGE clause)

7) Within a given report line, any two or more absolute items defined using column numbers that are not in increasing numerical order shall be subject to a different PRESENT WHEN clause.

8) The set of printable items within a given report line is subject to the following rules:

   a) If any two or more items overlap each other, they shall each be subject to a different PRESENT WHEN clause.

   b) The rightmost column positions of all absolute items shall not exceed the page width.

c) If the report line ends in a set of relative printable items or consists only of such, they shall not cause the page width to be exceeded unless each of them is subject to a different PRESENT WHEN clause, in which case this rule applies only to the largest of them.

9) If LEFT, CENTER, or RIGHT is specified, all the operands shall be absolute. If any of the operands is absolute and neither LEFT, CENTER, nor RIGHT is specified, LEFT is assumed.

10) If more than one integer-1 or integer-2 operand is specified the clause is referred to as a multiple COLUMN clause and the following additional rules apply:

a) No OCCURS clause shall be specified in the same entry.

b) All the occurrences of integer-1 shall be in increasing order of magnitude.

FORMAT 2

11) MINUS and – are synonyms.

12) Identifier-1 shall be described in the file, working-storage, local-storage, or linkage section as an elementary unsigned integer data item.

13) Neither the PLUS phrase nor the MINUS phrase shall be specified for the first elementary item in a screen record.

13.18.14.3 General rules

FORMAT 1

1) The COLUMN clause defines one or more printable items. If a COLUMN clause is not specified, the items are not printed.

2) There is a fixed correspondence, specified by the implementor, between a column and a character in a national character set.

3) The printable-size of a printable item is the number of columns required for printing the characters described by the item's PICTURE clause or, in the absence of a PICTURE clause, the literal specified in the VALUE clause. There is a one-to-one correspondence between a column and a character in an alphanumeric character set.

NOTE Columns might not line up in a report if the printable characters are not of a fixed size, such as in the character-set UTF-8.

4) Any report line shall be defined in such a way that any given column position is used for only one printable item when the line is printed. If this rule is violated the EC-REPORT-COLUMN-OVERLAP exception condition is set to exist and the results are undefined.

5) Any report line shall be defined in such a way that, when printed, the final column position of the last printable item does not exceed the page width. If this rule is violated the EC-REPORT-PAGE-WIDTH exception condition is set to exist, the report line is truncated, and the report line is printed.

6) If the integer-1 phrase is specified, the following general rules apply:

a) Integer-1 specifies an absolute column number.

b) If LEFT is specified, integer-1 is the leftmost column of the printable item. The rightmost column of the printable item is integer-1 + printable-size −1.

c) If RIGHT is specified, integer-1 is the rightmost column of the printable item. The leftmost column of the printable item is integer-1 −printable-size + 1.

d) If CENTER is specified, the printable item is centered, as follows:

1. If the printable-size of the printable item is an odd number of columns, integer-1 identifies the center column of the printable item. The leftmost column is integer-1 - ((printable-size - 1) / 2); the rightmost column is integer-1 + (printable-size / 2), truncated to an integer.

2. If the printable-size of the printable item is an even number of columns, integer-1 identifies the column of the printable item that is to the left of an imaginary line between the two middle columns of the printable item. The leftmost column is (integer-1 - (printable-size / 2)) + 1; the rightmost column is integer-1 + (printable-size / 2).

7) Within any given report line, a horizontal counter, representing the rightmost occupied column, is maintained and updated by each printable item in the line. At the start of the line, the horizontal counter is zero.

8) Integer-2 specifies a relative column number. If integer-2 is specified for an item that is the first printable item in the current line, it specifies the leftmost column of that item. Otherwise, the value of integer-2 is the number of column positions between the rightmost column of the preceding printable item and the leftmost column of the item being defined. The position of the item's leftmost character is obtained by adding integer-2 to the current line's horizontal counter.

NOTE The value of integer-2 minus 1 is the number of blank columns, if any, immediately preceding the item.

9) The rightmost column position of each printable item becomes the new value of the horizontal counter.

10) Any unoccupied columns in each print line are filled with space characters.

11) If an entry containing a LINE clause has no subordinate entry defining a printable item, the resultant report line will be blank.

12) A multiple COLUMN clause is functionally equivalent to a COLUMN clause with a single operand, together with a simple OCCURS clause whose integer is equal to the number of operands of the COLUMN clause, except that the multiple COLUMN clause allows the printable items to be defined at unequal horizontal intervals.

FORMAT 2

13) The COLUMN clause specifies the column in which the leftmost character of the screen item is to appear on the screen during the execution of an ACCEPT screen or a DISPLAY screen statement. Positioning of the screen record and within the screen record appears the same on the terminal display regardless of whether the whole screen record or just a portion of it is referenced in an ACCEPT screen or a DISPLAY screen statement.

NOTE Columns might not line up on a screen if the characters on the screen are not of a fixed size, such as in the character-set UTF-8.

14) If the COLUMN clause does not specify PLUS or MINUS, the clause gives the column number relative to the first column of the screen record. A column number of 1 represents the first column of the screen record.

15) If the PLUS or MINUS phrase is specified in the COLUMN clause, the column number is relative to the end of the preceding screen item in the same screen record, such that if COLUMN PLUS 1 is specified, the screen item starts immediately following the preceding screen item. PLUS denotes a column position that is increased by the value of identifier-1 or integer-3. MINUS denotes a column position that is decreased by the value of identifier-1 or integer-3.

16) A setting of COLUMN 1 is assumed for screen descriptions that specify the LINE clause but omit the COLUMN clause.

17) If both the LINE clause and the COLUMN clause are omitted, the following apply:

a) if no previous screen item has been defined, LINE 1 COLUMN 1 of the screen is assumed.

b) if a previous screen item has been defined, the line of that previous item and COLUMN PLUS 1 is assumed.

18) If a column number of zero is specified, the EC-SCREEN-STARTING-COLUMN exception condition is set to exist and the results are as if 1 were specified for the column number.

19) If the explicit or implicit column number of the first character of the screen item exceeds the number of columns available for a terminal, the EC-SCREEN-STARTING-COLUMN exception condition is set to exist and that screen item does not take part in the execution of the ACCEPT screen or DISPLAY screen statement.

Otherwise, if the starting column and length of a screen item are such that the field would extend beyond the end of the terminal line, the EC-SCREEN-ITEM-TRUNCATED exception condition is set to exist and the field is truncated at the end of the line.

13.18.15 CONSTANT RECORD clause

The CONSTANT RECORD clause identifies a structured constant. The content of a structured constant cannot be modified.

13.18.15.1 General format

CONSTANT RECORD

13.18.15.2 Syntax rules

1) The CONSTANT RECORD clause may be specified only in the local-storage or working-storage sections.

2) Neither the data item described by the subject of the entry nor any data item subordinate to the subject of the entry shall be specified as a receiving data item.

13.18.15.3 General rules

1) The content of a record described with the CONSTANT RECORD clause is as though the clause had been omitted and the record had been the subject of an INITIALIZE statement that is specified with the following phrases:

    — the FILLER phrase;
    — the VALUE phrase with the category-name of ALL;
    — and the DEFAULT phrase.

13.18.16 CONTROL clause

The CONTROL clause establishes a hierarchy of control breaks for the report.

13.18.16.1 General format

```
{ CONTROL IS  }  { { data-name-1 } ...          }
{ CONTROLS ARE } [ { FINAL [ data-name-1 ] ... } ]
```

13.18.16.2 Syntax rules

1) CONTROL and CONTROLS are synonyms.

2) Data-name-1 shall not be defined in the report section.

3) Data-name-1 may be qualified.

4) Data-name-1 may be reference modified. If it is, leftmost-position and length shall be integer literals.

5) The entry specified by data-name-1 shall not have an occurs-depending table subordinate to it.

6) Data-name-1 shall be unique in any given CONTROL clause. Two or more instances of data-name-1 in the same clause may, however, refer to the same physical data item or to overlapping data items.

7) Data-name-1 shall not reference a variable-length group.

13.18.16.3 General rules

1) The order of appearance of the operands of the CONTROL clause establishes the levels of the control hierarchy. The first data-name-1 is the major control; the next recurrence of data-name-1 is an intermediate control, etc. The last recurrence of data-name-1 is the minor control.

2) Specifying FINAL is equivalent to specifying a data item whose value does not change throughout the processing of the report. Therefore FINAL, if specified, is associated with the highest level in the hierarchy.

3) For each data-name-1 an internal data item, known as a prior control, is implicitly defined, having the same data description as the corresponding data item and a mechanism is established to save each control data item in the corresponding prior control and subsequently to compare these values to sense for control breaks. Execution of the chronologically first GENERATE statement for a given report saves each current control data item in the corresponding prior control. Subsequent executions of any GENERATE statement for that report automatically test the current value of each control data item, in order major to minor, for equality with the corresponding prior control. If a change of value in a control data item is detected, no further control data items are tested for the current GENERATE statement, and control break processing for that level and any lower levels is performed automatically.

4) If a control break has been detected during the execution of a GENERATE statement, the control data items are processed as follows:

   a) If any control footing is defined for the report, the current contents of control data items are saved and the corresponding prior controls are stored in the control data items before any control footing is printed. This ensures that any reference to a control data item in a control footing will always yield the value that the control data item had before the control break. If a USE BEFORE REPORTING declarative procedure is specified for the control footing, any reference to a control data item in this section will similarly yield the prior value. If the printing of the control footing causes a page advance and the page heading or page footing refers to a control data item, the value produced is similarly the prior value. When the printing of each control footing has been accomplished, the control data items are restored from the prior controls to

their new current values, with these new current values remaining in the prior controls. All subsequent references to the control data items will obtain the new current values.

b) If no control footing is defined for the report, the new current values of the control data items are stored in the corresponding prior controls.

(See also 13.18.56, TYPE clause.)

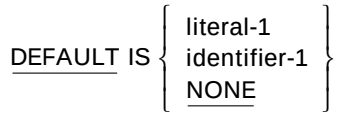
5) When a TERMINATE statement is executed, if any control footing is defined for the report, the prior controls are stored in the control data items before each control footing is printed, as though a highest-level control break had been detected. The result is the same as that of a control break detected during the execution of a GENERATE statement.

6) Data-name-1 shall not reference a zero-length group item.

13.18.17 DEFAULT clause

The DEFAULT clause specifies an explicit alternative value for a data item that contains all spaces or was found to be invalid on format during the format validation stage of the execution of a VALIDATE statement.

13.18.17.1 General format



13.18.17.2 Syntax rules

1) The subject of the entry shall not be of class index, object, or pointer.

2) Literal-1 or the data item referenced by identifier-1 shall be valid as a sending operand for a MOVE statement specifying the subject of the entry as the receiving operand.

13.18.17.3 General rules

1) A default value is established during the format validation stage of the execution of a VALIDATE statement if any of the following circumstances apply:

   a) In the case of an elementary data item, if the item's associated internal indicator is set to invalid on format, or the item is implicitly or explicitly of usage display or national and contains all spaces.

   b) In the case of an alphanumeric group or strongly-typed group item, if all the item's subordinate elementary data items are implicitly or explicitly of usage display and contain all spaces, and none of their associated internal indicators is set to invalid on format.

   c) In the case of a national group item, if all the item's subordinate elementary data items contain all spaces.

   In these cases, any subsequent reference to that data item during the execution of the current VALIDATE statement uses the default value rather than the actual content of the item. The data item itself remains unchanged. The DEFAULT clause is ignored during the execution of any statement other than a VALIDATE statement.

2) Except in the case of DEFAULT NONE, the DEFAULT clause specifies the default value explicitly. If literal-1 is specified, the default value used is the value of literal-1. If identifier-1 is specified, the default value used is the value of the data item referenced by identifier-1.

3) If the DEFAULT clause is not specified in the data item's data description entry, or if DEFAULT NONE is specified, the default value is that which would be supplied for the data item by the execution of an implicit INITIALIZE statement without a REPLACING or VALUE phrase.

4) If NONE is specified, the effect of the clause is the same as if no DEFAULT clause were specified, except that, if the data item contains all spaces or the data item is elementary and is found to be invalid on format, any DESTINATION clause specified for the data item is ignored.

13.18.18 DESTINATION clause

The DESTINATION clause specifies one or more data items to which data is to be implicitly moved during the execution of the input distribution stage of a VALIDATE statement.

13.18.18.1 General format

DESTINATION IS { identifier-1 } ...

13.18.18.2 Syntax rules

1) The description of the data item referenced by identifier-1 shall be such that a syntactically correct MOVE statement results when that data item is the receiving operand and the subject of the entry, qualified and subscripted where necessary, is the sending operand.

2) The data description entry for the data item referenced by identifier-1 or any data item subordinate to identifier-1 shall not contain a VALIDATE-STATUS clause.

3) If the subject of the entry is a global name, every data-name appearing in identifier-1 shall be a global name.

13.18.18.3 General rules

1) The DESTINATION clause causes one or more implicit MOVE statements to be executed during the input distribution phase of the execution of a VALIDATE statement that references a data item whose description contains the DESTINATION clause in the same or in a subordinate entry. The DESTINATION clause is ignored during the execution of any statement other than a VALIDATE statement.

2) If a default value has not been assigned to the item, as described in general rule 1 of the DEFAULT clause, the actual content of the item is moved to the data item referenced by identifier-1.

3) If a default value has been assigned to the item and DEFAULT NONE is not specified in the entry, the item's default value is moved to the data item referenced by identifier-1.

4) If a default value has been assigned to the item and DEFAULT NONE is specified in the entry, the data item referenced by identifier-1 is unchanged.

5) If identifier-1 contains subscripts defined in VARYING clauses that are processed at the same time as the DESTINATION clause, these subscripts may be used to store different occurrences of the data item in the same number of different occurrences of the destination item.

   NOTE If a VARYING clause is not used, any subscripts specified in the DESTINATION clause are unchanged.

6) The sending data determined by the above general rules is moved to the data item referenced by each identifier-1 in the order specified, according to the rules for the MOVE statement.

13.18.19 Entry-name clause

The entry-name clause specifies the name of the item that is being described. A data-name or screen-name may be used to give the item a name.

13.18.19.1 General format

Format 1 (data-name)

data-name-1

Format 2 (screen-name)

screen-name-1

Format 3 (filler)

FILLER

13.18.19.2 Syntax rules

1) If the entry-name clause is not specified in a data description entry, a report group description entry, or a screen description entry, the word FILLER is assumed.

2) If the entry-name clause is specified in a data description entry or a report group description entry, either data-name-1 or FILLER shall be specified. If the entry-name clause is specified in a screen description entry, either screen-name-1 or FILLER shall be specified.

13.18.19.3 General rules

1) The word FILLER may be used to name a data, report, or screen item. Under no circumstances shall a FILLER item be referred to explicitly.

13.18.20 ERASE clause

The ERASE clause clears part of the line or the screen starting at the cursor position.

13.18.20.1 General format

```
ERASE { END OF LINE    }
      { END OF SCREEN  }
      { EOL            }
      { EOS            }
```

13.18.20.2 Syntax rules

1) The word EOL is equivalent to the words END OF LINE.

2) The word EOS is equivalent to the words END OF SCREEN.

13.18.20.3 General rules

1) When the ERASE clause is specified, a portion of the screen is cleared during the execution of a DISPLAY screen statement before data is transferred to the screen item. The clearing begins at the line and column coordinates specified for the subject of the entry and continues as follows:

   — If LINE is specified, the clearing continues to the end of the line.

   — If SCREEN is specified, the clearing continues to the end of the screen.

2) The ERASE clause is ignored during all phases of the execution of an ACCEPT screen statement.

13.18.21 EXTERNAL clause

The EXTERNAL clause specifies that a data item or a file connector is external. The constituent data items and group data items of an external data record are available in a run unit to every runtime element that describes the record as external.

13.18.21.1 General format

IS EXTERNAL [AS literal-1]

13.18.21.2 Syntax rules

1) The EXTERNAL clause may be specified only in file description entries or in level 1 data description entries in the working-storage section.

2) In the same source element, the externalized name of the subject of the entry that includes the EXTERNAL clause shall not be the same as the externalized name of any other entry that includes the EXTERNAL clause.

3) Literal-1 shall be an alphanumeric or national literal and shall be neither a figurative constant nor a zero-length literal.

4) The EXTERNAL clause shall not be specified for a data item of class object or pointer or for a strongly-typed group item.

13.18.21.3 General rules

1) If the EXTERNAL clause is specified in a record description entry, the data contained in the record is external and may be accessed within the run unit by any runtime element that describes the same record as external, subject to the following rules.

   NOTE Use of the EXTERNAL clause does not imply that the associated file-name or data-name is a global name.

2) If the EXTERNAL clause is specified in a file description entry:

   a) the file connector associated with this file description entry is an external file connector; and

   b) the data contained in all record description entries subordinate to that file description entry is external and may be accessed by any runtime element in the run unit that describes the same file and records as external, subject to the following rules.

3) Literal-1, if specified, is the name of the file connector or record that is externalized to the operating environment. If literal-1 is not specified, the externalized name of the file connector or record is the name specified in the file description entry or the data-name format of the entry-name clause, respectively.

4) Within a run unit, if two or more source elements describe the same external data record, each name that is externalized to the operating environment for the record description entries shall be the same; the VALUE clause specification, if any, for each record name of the associated record description entries shall be identical; and the records shall define the same number of bytes. A source element that describes an external record may contain a data description entry including the REDEFINES clause that redefines the complete external record, and this complete redefinition need not occur identically in other source elements in the run unit.

13.18.22 FOREGROUND-COLOR clause

The FOREGROUND clause specifies the foreground color for the screen item.

13.18.22.1 General format

FOREGROUND-COLOR IS { identifier-1 / integer-1 }

13.18.22.2 Syntax rules

1) Identifier-1 shall be described in the file, working-storage, local-storage, or linkage section as an unsigned integer data item.

2) Integer-1 shall have a value in the range of 0 to 7.

13.18.22.3 General rules

1) When an ACCEPT screen statement or a DISPLAY screen statement referencing the associated screen item is executed, the content of the data item referenced by identifier-1 shall be in the range of 0 to 7.

2) Integer-1 or the content of the data item referenced by identifier-1 specifies the color number of the foreground color to be used for displaying the screen item. The color associated with the color number is specified in 9.2.7, Color number.

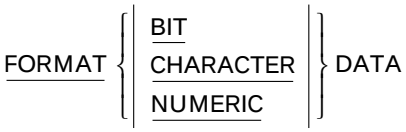
3) A FOREGROUND-COLOR clause specified at the group level applies to each elementary screen item in that group.

4) When a FOREGROUND-COLOR clause is not specified or the value is not in the range 0 to 7, the foreground color is implementor-defined.

13.18.23 FORMAT clause

The FORMAT clause specifies that records written to the file are to be formatted to an external representation during the output process and records read from the file are to be converted from the external format into internal representation during the input process. The external representation is suitable for presentation or printing.

13.18.23.1 General format



13.18.23.2 Syntax rules

1) If more than one record description entry is associated with a file description entry in which the FORMAT clause is specified, each of these record description entries shall contain a SELECT WHEN clause.

2) The REDEFINES clause and the RENAMES clause shall not be specified in a record description entry associated with a file description entry in which a FORMAT clause is specified or in the data description entry of the identifier specified in a FROM phrase in a REWRITE or WRITE statement referencing that file description entry.

3) Data items of class index, object, and pointer shall not be specified in a record description entry associated with a file description entry in which a FORMAT clause is specified or in the data description entry of the identifier specified in a FROM phrase in a REWRITE or WRITE statement referencing that file description entry.

13.18.23.3 General rules

1) The FORMAT clause specifies that external media format is to be used for records written to the file. In external media format, data contains the appropriate encoding for presentation or printing.

   NOTE The data in the record area is visible to the runtime module in internal representation.

2) A record description entry or a data description entry is selected for use in formatting during an input-output operation referencing a file description entry in which a FORMAT clause is specified as follows:

   a) If only one record description entry is associated with the file description entry, that record description entry is selected.

   b) If more than one record description entry is associated with the file description entry, the record description entry identified by the SELECT WHEN clause is selected.

   c) If there are no record description entries associated with the file description entry, the record description entry or data description entry that is selected depends on the input-output operation as follows:

      1. If the input-output statement is a READ statement, it is the data description entry of the identifier that is specified in the INTO phrase of that READ statement.

      2. If the input-output statement is a WRITE or REWRITE statement and an identifier is specified in the FROM phrase, it is the data description entry of that identifier.

      3. If the input-output statement is a WRITE or REWRITE statement and a literal is specified in the FROM phrase, an implicit data description entry that describes the literal is the data description entry as follows:

a. If the literal is an alphanumeric literal, the implicit data description entry is

```
01 PIC X(n).
```

where n is the number of alphanumeric character positions in the literal.

b. If the literal is a boolean literal, the implicit data description entry is

```
01 PIC 1(n).
```

where n is the number of boolean character positions in the literal.

c. If the literal is a national literal, the implicit data description entry is

```
01 PIC N(n).
```

where n is the number of national character positions in the literal.

3) Elementary data description entries of the selected record description entry are used to determine the data items to be formatted.

4) If CHARACTER is specified, data items described with the following characteristics are selected for formatting:

— class alphabetic
— class alphanumeric
— class boolean with usage display
— class national
— class numeric with usage display
— class numeric with usage national.

5) If BIT is specified, data items described with class boolean and usage bit are selected for formatting.

6) If NUMERIC is specified, data items described with class numeric and a usage other than national or display are selected for formatting.

7) If a data item is not selected for formatting, it is transferred as is with no conversion or formatting taking place.

8) For a WRITE or REWRITE statement, each selected data item is formatted as necessary in a temporary area for output in external media format. Output formatting does not appear in the record area. Signs are presented as SIGN IS LEADING SEPARATE.

9) For a READ statement, each selected data item is formatted as necessary to the appropriate internal representation. The record area contains the resultant logical record in internal representation.

10) The implementor shall specify the representation produced as a result of processing the FORMAT clause, including any control information that may be necessary for presentation or printing. This representation may be the same as internal representation.

11) Data items selected for formatting are restored to the same internal representation when read with the same FORMAT clause on the same implementation, except for exclusions specified by the implementor.

12) If the associated file connector is an external file connector, all file description entries in the run unit that are associated with that file connector shall specify the same FORMAT clause.

13.18.24 FROM clause

The FROM clause specifies the source of data for an ACCEPT screen statement and a DISPLAY screen statement.

13.18.24.1 General format

FROM { identifier-1 / literal-1 }

13.18.24.2 Syntax rules

1) Identifier-1 shall be defined in the file, working-storage, local-storage, or linkage section.

2) The category of identifier-1 and literal-1 shall be a permissible category as a sending operand in a MOVE statement where the receiving operand has the same PICTURE clause as the subject of the entry.

3) If the subject of this entry is subject to an OCCURS clause, identifier-1 shall be specified without the subscripting normally required. Additional requirements are specified in 13.18.37, OCCURS clause, syntax rule 13.

4) Identifier-1 shall not specify a variable-length group.

5) Literal-1 shall not be a zero-length literal.

13.18.24.3 General rules

1) The subject of the entry is an output screen item.

13.18.25 FULL clause

The FULL clause specifies that the operator shall either leave the screen item completely empty or fill it entirely with data.

13.18.25.1 General format

FULL

13.18.25.2 General rules

1) If a FULL clause is specified at the group level, it applies to each elementary input screen item in that group that does not have the JUSTIFIED clause specified.

2) The FULL clause is effective during the execution of any ACCEPT screen statement that causes the screen item to be accepted, provided the cursor enters the screen item at some time during the execution of the ACCEPT statement.

3) The effect of the FULL clause is to reject the normal termination key unless that clause is satisfied. To satisfy the clause for a screen item:

   a) If it is alphanumeric or alphanumeric-edited, either the entire item shall contain spaces, or both the first and last character positions shall contain nonspace characters;

   b) If it is national or national-edited, either the entire item shall contain spaces, or both the first and last character positions shall contain nonspace characters;

   c) If it is numeric or numeric-edited, either the value shall be zero, or there shall be no digit position in which zero suppression has taken effect;

   d) If it is boolean, the entire item shall contain boolean '0' or '1' characters.

4) For fields that are both input and output, the FULL clause may be satisfied by the contents of the literal or identifier referenced in the FROM or USING clause, as well as operator keyed data.

5) The FULL clause shall not be effective if a function key is used to terminate the execution of the ACCEPT statement.

6) The specification of the FULL and REQUIRED clauses together requires that the field be entirely filled before the normal termination key has any effect.

7) The FULL clause is ignored for a field that is not an input field.

13.18.26 GLOBAL clause

The GLOBAL clause specifies that a constant-name, a data-name, a file-name, a report-name, or a screen-name is a global name. A global name is available to every program contained within the program that declares it.

13.18.26.1 General format

IS GLOBAL

13.18.26.2 Syntax rules

1) The GLOBAL clause may be specified only in the following entries:

   a) A constant entry.

   b) A data description entry whose level-number is 1 that is specified in the file, working-storage, local-storage, or linkage section.

   c) A screen-description entry whose level-number is 1.

   d) A file description entry.

   e) A report description entry.

2) If the SAME RECORD AREA clause is specified for several files, the record description entries or the file description entries for these files shall not include the GLOBAL clause.

3) If the GLOBAL clause is not specified in the file description entry of a containing program, the file shall not be referenced directly or indirectly by any input-output statements in any contained program.

4) The GLOBAL clause shall not be specified in a factory definition, an instance definition, or a method definition.

13.18.26.3 General rules

1) A constant-name, data-name, file-name, report-name, or screen-name described using a GLOBAL clause is a global name. All data-names subordinate to a global name are global names. All condition-names associated with a global name are global names.

2) A statement in a program contained directly or indirectly within a program that describes a global name may reference that name without describing it again. (See 8.4.5, Scope of names.)

3) If the GLOBAL clause is used in a data description entry that contains the REDEFINES clause, it is only the subject of that REDEFINES clause that possesses the global attribute.

13.18.27 GROUP INDICATE clause

The GROUP INDICATE clause specifies that the associated printable item is printed only on the first occurrence of its report group after execution of an INITIATE statement, or after a control break or page advance.

13.18.27.1 General format

GROUP INDICATE

13.18.27.2 Syntax rules

1) The GROUP INDICATE clause may be specified only within a detail report group description, in an elementary entry that also contains a COLUMN clause and a SOURCE or VALUE clause.

13.18.27.3 General rules

1) The GROUP INDICATE clause has the same effect as a PRESENT WHEN clause where the associated condition is true only on the first occasion that a GENERATE is issued for the current detail group after any of the following events. (See 13.18.40, PRESENT WHEN clause.)

    a) Execution of an INITIATE for the report.

    b) A page advance, if the report is divided into pages. (See 13.18.38, PAGE clause.)

    c) Any control break, if the report description contains a CONTROL clause.

13.18.28 GROUP-USAGE clause

A GROUP-USAGE clause with a BIT phrase specifies that the group item defined by the subject of the entry is to be treated as an elementary item of usage bit and category boolean, unless specified otherwise.

A GROUP-USAGE clause with a NATIONAL phrase specifies that the group item defined by the subject of the entry is to be treated as an elementary item of usage national and category national, unless otherwise specified.

13.18.28.1 General format

GROUP-USAGE IS { BIT | NATIONAL }

13.18.28.2 Syntax rules

1) The GROUP-USAGE clause may be specified only if the subject of the entry is a group item that is not strongly typed and not a variable-length group.

2) When the BIT phrase is specified, USAGE BIT is implied for the subject of the entry. A USAGE clause shall not be explicitly specified for the subject of the entry, but USAGE BIT may be implicitly specified. All elementary items subordinate to the subject of the entry shall be explicitly or implicitly described as usage bit, class and category boolean. All subordinate group items shall be explicitly or implicitly described as GROUP-USAGE BIT.

3) When the NATIONAL phrase is specified, USAGE NATIONAL is implied for the subject of the entry. A USAGE clause shall not be explicitly specified for the subject of the entry, but USAGE NATIONAL may be implicitly specified. All elementary items subordinate to the subject of the entry shall be explicitly or implicitly described as usage national. Any signed numeric data items shall be described with the SIGN IS SEPARATE clause. All subordinate group items shall be explicitly or implicitly described as GROUP-USAGE NATIONAL.

13.18.28.3 General rules

1) When the BIT phrase is specified:

   a) The subject of the entry is a bit group and also a bit data item; its class and category are boolean.

   b) Unless stated otherwise, a bit group is treated as though it were an elementary data item of usage bit and class and category boolean described with PICTURE 1(m), where m is the bit length of the group.

   c) Data items contained within a bit group are allocated in storage in accordance with the rules specified in 8.5.1.5.3, Alignment of data items of usage bit.

2) When the NATIONAL phrase is specified:

   a) The subject of the entry is a national group; its class and category are national.

   b) Unless stated otherwise, a national group is treated as though it were an elementary data item of usage national and class and category national described with PICTURE N(m), where m is the length of the group.

      NOTE The GROUP-USAGE NATIONAL clause is needed so that groups containing only national characters can be properly truncated and padded with national characters and can be correctly processed for operations such as INSPECT. Without the GROUP-USAGE NATIONAL clause, the content of such a group item would be treated as category alphanumeric, possibly leading to corruption or invalid handling of data.

3) If a GROUP-USAGE clause is not specified or implied for a group item that is not strongly typed and is not a variable-length group, that group item is an alphanumeric group item.

13.18.29 HIGHLIGHT clause

The HIGHLIGHT clause specifies that the field is to appear on the screen with the highest level of intensity.

13.18.29.1 General format

HIGHLIGHT

13.18.29.2 General rules

1) A HIGHLIGHT clause specified at the group level applies to each elementary screen item in that group.

2) When the HIGHLIGHT clause is specified, the characters that constitute the screen item will be displayed in the foreground color at the highest intensity when the screen item is referenced in an ACCEPT screen or a DISPLAY screen statement.

13.18.30 INVALID clause

The INVALID clause specifies the circumstances under which a data item fails the relation validation stage of the execution of a VALIDATE statement.

13.18.30.1 General format

INVALID WHEN condition-1

13.18.30.2 Syntax rules

1) The subject of the entry shall not be of class index, object, or pointer.

13.18.30.3 General rules

1) The INVALID clause takes effect during the relation validation stage of the execution of a VALIDATE statement. The INVALID clause is ignored during the execution of any statement other than a VALIDATE statement.

2) The INVALID clause takes effect only if the item's associated internal indicator is still set to its initial valid value.

3) Condition-1 is evaluated at the beginning of the execution of the relation validation stage, with the following two possible results:

    a) If condition-1 is true, the item's associated internal indicator is set to invalid on relation.

    b) If condition-1 is false, the INVALID clause has no effect.

13.18.31 JUSTIFIED clause

The JUSTIFIED clause specifies right justification of data within a receiving data item or screen item.

13.18.31.1 General format

```
{ JUSTIFIED } RIGHT
{ JUST      }
```

13.18.31.2 Syntax rules

1) The JUSTIFIED clause may be specified only at the elementary item level.

2) JUST is an abbreviation for JUSTIFIED.

3) The JUSTIFIED clause may be specified only for a data item whose category is alphabetic, alphanumeric, boolean, or national.

4) The JUSTIFIED clause shall not be specified for an any-length elementary item.

13.18.31.3 General rules

1) When the receiving data item is described with the JUSTIFIED clause and the sending operand is larger than the receiving data item, the leftmost character positions or boolean positions of the sending operand shall be truncated.

2) When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending operand, the data is aligned at the rightmost character position or boolean position in the data item with zero fill for the leftmost boolean positions and space fill for the leftmost character positions. For data items implicitly or explicitly described as usage national, national zeros shall be used for zero fill and national spaces shall be used for space fill. For data items implicitly or explicitly described as usage display, alphanumeric zeros shall be used for zero fill and alphanumeric spaces shall be used for space fill. For data items implicitly or explicitly described as usage bit, bit zeros shall be used for zero fill.

3) When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply (see 14.6.8, Alignment of data within data items).

13.18.32 Level-number

Level numbers 1 through 49 indicate the position of a data item or screen item within the hierarchical structure described by a data description entry, a report group description entry, or a screen description entry. In addition, level numbers 66, 77, and 88 are used to identify special entries.

13.18.32.1 General format

level-number

13.18.32.2 Syntax rules

1) A level-number is required as the first element in each data description or screen description entry.

2) Data description entries subordinate to a FD or SD entry shall have level-numbers with the values 66, 88, or 1 through 49.

3) A level-number in the range of 1 through 9 may be specified as 01 through 09.

4) Report group description entries that are subordinate to an RD entry shall have level-numbers with the values 1 through 49.

5) Data description entries in the working-storage section, local-storage section, and linkage section shall have level-numbers 66, 77, 88, or 1 through 49.

6) Screen description entries shall have level-numbers 1 through 49.

13.18.32.3 General rules

1) The level-number 1 identifies the first entry in each record description, type declaration, or report group.

2) Special level-numbers have been assigned to certain entries where there is no real concept of hierarchy:

   a) Level-number 77 is assigned to identify noncontiguous working storage data items, noncontiguous local storage data items, and noncontiguous linkage data items, and may be used only as described by the data description format of the data description entry.

   b) Level-number 66 is assigned to identify RENAMES entries and may be used only as described by the renames format of the data description entry.

   c) Level-number 88 is assigned to entries that define condition-names associated with a conditional variable and to define criteria to be used to validate a data item. Level-number 88 may be used only as described by the condition-name format or the validation format of the data description entry.

3) Multiple level 1 entries subordinate to a FD or SD entry represent implicit redefinitions of the same area. Multiple level 1 entries subordinate to a report description entry do not represent implicit redefinitions of the same area.

13.18.33 LINAGE clause

The LINAGE clause provides a means for specifying the depth of a logical page in terms of number of lines. It also provides for specifying the size of the top and bottom margins on the logical page, and the line number, within the page body, at which the footing area begins.

13.18.33.1 General format

```
LINAGE IS { data-name-1 } LINES [ WITH FOOTING AT { data-name-2 } ]
          { integer-1   }                         { integer-2   }

[ LINES AT TOP { data-name-3 } ] [ LINES AT BOTTOM { data-name-4 } ]
               { integer-3   }                     { integer-4   }
```

13.18.33.2 Syntax rules

1) Data-name-1, data-name-2, data-name-3, data-name-4 may be qualified and shall reference elementary unsigned numeric integer data items.

2) Integer-2 shall not be greater than integer-1.

3) Integer-3, integer-4 may be zero.

13.18.33.3 General rules

1) The LINAGE clause provides a means for specifying the size of a logical page in terms of number of lines. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the values of these items are zero. If the FOOTING phrase is not specified, no end-of-page condition independent of the page overflow condition exists.

   There is not necessarily any relationship between the size of the logical page and the size of a physical page.

2) Integer-1 or the value of the data item referenced by data-name-1 specifies the number of lines that may be written or spaced on the logical page. This number is called the page size. That part of the logical page in which these lines may be written or spaced is called the page body.

3) Integer-2 or the value of the data item referenced by data-name-2 specifies the line number within the page body at which the footing area begins. The value shall be greater than zero and not greater than integer-1 or the value of the data item referenced by data-name-1.

   The footing area is the area of the page body between the footing start and the page size, inclusive.

4) Integer-3 or the value of the data item referenced by data-name-3 specifies the number of lines in the top margin on the logical page. This number is called the top margin.

5) Integer-4 or the value of the data item referenced by data-name-4 specifies the number of lines in the bottom margin on the logical page. This number is called the bottom margin.

6) The values for the page size, top margin, footing start, and bottom margin are determined as follows:

   a) If a literal is specified, the value is always that literal.

   b) If a data-name is specified, the value is the content of the data item referenced by the associated data-name at the following times when the indicated statement references the associated file:

1. At the completion of an OPEN statement with the OUTPUT phrase.

2. During the execution of a WRITE statement that is specified with the ADVANCING PAGE phrase. This occurs before the device is positioned and after all positioning on the current page.

3. During the execution of a WRITE statement that causes a page overflow condition. This occurs before the device is positioned and after all positioning on the current page.

When a value is determined for the page size, top margin, footing start, and bottom margin, the value applies to the next logical page.

The value shall conform to the following rules:

1. The page size shall be greater than zero.

2. The footing start shall be greater than zero and not greater than the page size.

If the value does not conform to these two rules, the EC-I-O-LINAGE exception condition is set to exist. If execution continues after processing of this exception condition, it continues with the statement following the WRITE statement; the LINAGE -COUNTER is set to 0 and remains at that value until the file is closed; and all subsequent WRITE statements referencing the file cause the EC-I-O-LINAGE exception condition to continue to exist until the file is closed.

7) A LINAGE-COUNTER is generated by the presence of a LINAGE clause. The value in the LINAGE-COUNTER at any given time represents the line number at which the device is positioned within the current page body. The rules governing the LINAGE-COUNTER are as follows:

   a) A separate LINAGE-COUNTER is supplied for each file described in the file section whose file description entry contains a LINAGE clause.

   b) LINAGE-COUNTER may be referenced only in procedure division statements; however only the input-output control system may change the value of LINAGE-COUNTER. Since more than one LINAGE-COUNTER may exist in a source element, the user shall qualify LINAGE-COUNTER by file-name when necessary.

   c) LINAGE-COUNTER is automatically modified, according to the following rules, during the execution of a WRITE statement to an associated file:

      1. When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to one. During the resetting of LINAGE-COUNTER to the value one, the value of LINAGE-COUNTER is implicitly incremented to exceed the value specified by integer-1 or the data item referenced by data-name-1.

      2. When the ADVANCING phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by the value of the integer specified in the ADVANCING phrase or the contents of the data item referenced by the identifier specified in the ADVANCING phrase.

      3. When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value one.

      4. The value of LINAGE-COUNTER is automatically reset to one when the device is repositioned to the first line that may be written on for each of the succeeding logical pages.

   d) The value of LINAGE-COUNTER is automatically set to one at the time an OPEN statement with the OUTPUT phrase is executed for the associated file.

8) Each logical page is contiguous to the next with no additional spacing provided.

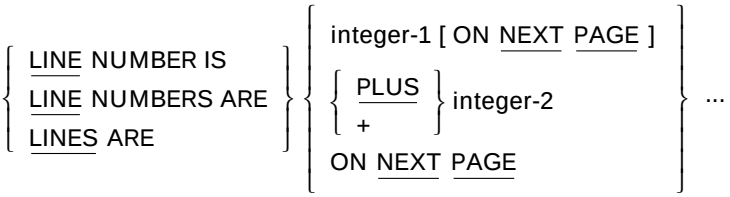
13.18.34 LINE clause

The LINE clause introduces a report line, or a set of adjacent report lines, and specifies their vertical positioning.

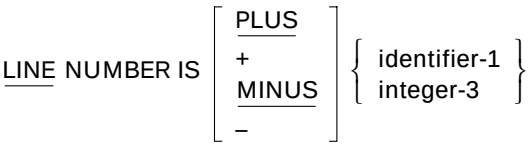
The LINE clause also specifies vertical positioning for its screen item.

13.18.34.1 General format

Format 1 (report-writer):



Format 2 (screen):



13.18.34.2 Syntax rules

ALL FORMATS

1) PLUS and + are synonyms.

FORMAT 1

2) LINE and LINES are synonyms.

3) Integer-1 specifies an absolute line number. Integer-2 specifies a relative line number. Neither integer-1 nor integer-2 shall exceed the page limit, or 9999 if the report is not divided into pages. (See 13.18.38, PAGE clause.) Integer-2 may be zero. Integer-1 and integer-2 shall be unsigned.

4) Within a given report group description entry, an entry that contains a LINE clause shall not have a subordinate entry that also contains a LINE clause.

5) If the report is not divided into pages, all its LINE clauses shall be relative.

6) Within any given report group, the set of report lines, if any, is subject to the following rules:

   a) If any two or more absolute lines are defined using line numbers that are not in increasing numerical order, they shall each be subject to a different PRESENT WHEN clause.

   b) If any two or more lines, or groups of lines, overlap each other, they shall each be subject to a different PRESENT WHEN clause.

   c) Any absolute report lines shall be defined in such a way that no line appears above the upper limit or below the lower limit allowed for the report group. The upper and lower limits for each type of report group are given in the general rules for the TYPE clause. (See 13.18.56, TYPE clause.)

d) If the report group consists of one or more absolute lines, not subject to any PRESENT WHEN clause, and ends in a set of relative lines, or groups of relative lines, they shall not cause the report group's lower limit to be exceeded unless each of them is subject to a different PRESENT WHEN clause, in which case this rule applies only to the vertically largest of them.

e) If the description of any absolute line appears later than that of a relative line, they shall each be subject to a different PRESENT WHEN clause.

7) Within a given report group description, a NEXT PAGE phrase, if present, shall be specified only in the first LINE clause.

8) The NEXT PAGE phrase may appear only in the description of a body group or a report footing.

9) If the current report group is a control heading with the OR PAGE phrase, all the LINE clauses in the report group description shall be relative.

10) If more than one integer-1 or integer-2 operand is specified, the clause is referred to as a multiple LINE clause and the following additional rules apply:

a) The NEXT PAGE phrase, if specified, shall appear only with the first operand.

b) All absolute operands, if present, shall precede all relative operands, if present.

c) The occurrences of integer-1, if present, shall be in ascending numerical order.

d) An OCCURS clause shall not also be present in the same entry.

FORMAT 2

11) MINUS and – are synonyms.

12) Identifier-1 shall be described in the file, working-storage, local-storage, or linkage section as an elementary unsigned integer data item.

13) Neither the PLUS phrase nor the MINUS phrase shall be specified for the first elementary item in a screen record.

13.18.34.3 General rules

FORMAT 1

1) Each LINE clause in a given report group description defines a report line, or set of report lines, that is printed when the report group is printed. The report's LINE-COUNTER identifier contains the line number within the page of the most recent line to have been printed.

2) If the report is divided into pages, each report group shall be described such that it may be printed on one page. A report group shall never be split between two pages. If this rule is violated the EC-REPORT-PAGE-LIMIT exception condition is set to exist and the results are undefined.

3) Each report group shall be described such that, when it is printed, no lines or groups of lines overlap each other, except that the nonspace characters of a relative line specified with an integer-2 of zero will overwrite the corresponding characters of the preceding line. If this rule is violated the EC-REPORT-LINE-OVERLAP exception condition is set to exist and the results are undefined.

4) If the report is divided into pages and the report group about to be printed is a body group, other than the chronologically first body group since the execution of an INITIATE statement for the report, a page fit test is performed before any line of the report group is printed. The rules for the page fit test and the location of the first line of the body group are modified if the immediately preceding body group's description contained the

absolute form of a NEXT GROUP clause. (See 13.18.36, NEXT GROUP clause.) The actions described below assume that no such NEXT GROUP clause was present.

The nature of the page fit test depends on the format of the first LINE clause specified in the report group description. Which LINE clause is taken to be the first may depend on the current values of conditions in PRESENT WHEN clauses in the report group description. (See 13.18.40, PRESENT WHEN clause.)

The page fit test takes one of the following forms:

a) If the first LINE clause has a NEXT PAGE phrase, no page fit test takes place and the page fit is declared unsuccessful.

b) If the first LINE clause is absolute without the NEXT PAGE phrase, integer-1 (the target position of the first line) is compared with the current value of LINE-COUNTER. If integer-1 is greater than LINE-COUNTER, the page fit is declared successful; otherwise the page fit is declared unsuccessful.

c) If the first LINE clause is relative (so that all LINE clauses in the report group description are relative) without the NEXT PAGE phrase, the expected position of the last line of the report group is computed in a trial sum. First, the current value of LINE-COUNTER is placed in the trial sum. Next, the trial sum is incremented by integer-2 for each subsequent LINE clause. Wherever there is an OCCURS clause at the level of the LINE clause or above, the vertical interval between successive occurrences is added into the trial sum once for each occurrence beyond the first. (See 13.18.37, OCCURS clause.)

   If any of the LINE clauses used in computing the trial sum are subject to a PRESENT WHEN clause or to an OCCURS clause with the DEPENDING phrase, these clauses are taken into account in computing the trial sum. (See 13.18.40, PRESENT WHEN clause.)

   If the trial sum is less than or equal to the lower limit for the report group, the page fit is declared successful; otherwise the page fit is declared unsuccessful. The lower limit for body groups is defined in the general rules under 13.18.56, TYPE clause.

If the page fit was declared successful, no page advance takes place and the report group is printed on the current page; otherwise, a page advance takes place before the body group is printed and the body group becomes the first body group on the new page as specified in 14.9.15, GENERATE statement.

5) The line number on which the report group's first line is printed depends on the first LINE clause in the report group description. Which LINE clause is taken to be the first may depend on the current values of conditions in PRESENT WHEN clauses in the report group description. (See 13.18.40, PRESENT WHEN clause.)

   a) If the first LINE clause of the report group is absolute, the report group's first line number is given by integer-1. If in addition the NEXT PAGE phrase is present and the report group is a report footing, the first line is printed beginning on a new page.

   b) If the first LINE clause of the report group is relative and the report is divided into pages, the report group's first line number is calculated as follows:

      1. If the report group is a report heading, the line number is given by (HEADING integer + integer-2 - 1).

      2. If the report group is a page heading and no report heading has been printed on the same page, the line number is given by (HEADING integer + integer-2 - 1). If the report group is a page heading and a report heading has been printed on the same page, the line number is given by adding integer-2 to the current value of the report's LINE-COUNTER.

      3. When the report group is a body group: if that report group is the first body group to be printed on the current page, the line number is given by the FIRST DETAIL integer; otherwise, the line number is given by adding integer-2 to the current value of the report's LINE-COUNTER.

      4. If the report group is a page footing, the line number is given by (FOOTING integer + integer-2).

5. When the report group is a report footing: if no page footing has been printed on the same page, the line number is given by (FOOTING integer + integer-2); otherwise, the line number is given by adding integer-2 to the current value of the report's LINE-COUNTER.

c) If the first LINE NUMBER clause of the report group is relative and the report is not divided into pages, the report group's first line number is obtained by adding integer-2 to the current value of the report's LINE-COUNTER.

6) When the first line number of the report group has been calculated, the report's LINE-COUNTER is set equal to that line number and the line is now printed on the page at that vertical location.

7) Any subsequent LINE clauses within the same report group cause the line number to be updated as follows:

a) If the LINE clause is absolute, the new line number is given by integer-1.

b) If the LINE clause is relative, the new line number is determined by adding integer-2 to the report's LINE-COUNTER.

When each subsequent line number of the report group has been calculated, the report's LINE-COUNTER is set equal to that number and the line is now printed on the page at that vertical location.

Line numbers need not be consecutive. Any unoccupied lines on the page result in a blank line.

8) The final line number for the report group is given by the value contained in the report's LINE-COUNTER when the last line of the report group has been printed. Before processing of the report group is concluded, the value of LINE-COUNTER may be further altered as a result of the action of an optional NEXT GROUP clause defined for the report group. (See 13.18.36, NEXT GROUP clause.)

9) A multiple LINE clause is functionally equivalent to a LINE clause with a single operand, together with a simple OCCURS clause whose integer is equal to the number of operands of the LINE clause, except that the multiple LINE clause allows the report lines to be defined at unequal vertical intervals.

FORMAT 2

10) The LINE clause, in conjunction with the COLUMN clause, establishes the starting coordinates for a screen item within a screen record. The LINE clause specifies the vertical coordinate. Positioning of the screen record and within the screen record appears the same on the terminal display regardless of whether the whole screen record or just a portion of it is referenced in an ACCEPT screen or a DISPLAY screen statement.

11) If the LINE clause does not specify PLUS or MINUS, the clause gives the line number relative to the start of the screen record. A line number of 1 represents the first line of the screen record.

12) If the PLUS or MINUS phrase is specified in the LINE clause, the line number is relative to the end of the preceding screen item in the same screen record. PLUS denotes a line position that is increased by the value of identifier-1 or integer-3. MINUS denotes a line position that is decreased by the value of identifier-1 or integer-3.

13) If the LINE clause is omitted, the following apply:

a) if no previous screen item has been defined, LINE 1 of the screen record is assumed.

b) if a previous screen item has been defined, the line of that previous item is assumed.

14) If the explicit or implicit line number of a screen item is zero or exceeds the number of lines available for a terminal, the EC-SCREEN-LINE-NUMBER exception condition is set to exist and that screen item does not take part in the execution of the ACCEPT screen or DISPLAY screen statement.

13.18.35 LOWLIGHT clause

The LOWLIGHT clause specifies that the field is to appear on the screen with the lowest level of intensity.

13.18.35.1 General format

LOWLIGHT

13.18.35.2 General rules

1) A LOWLIGHT clause specified at the group level applies to each elementary screen item in that group.

2) When the LOWLIGHT clause is specified, the characters that constitute the screen item will be displayed in the foreground color at the lowest intensity when the screen item is referenced in an ACCEPT screen or a DISPLAY screen statement.

13.18.36 NEXT GROUP clause

The NEXT GROUP clause specifies additional blank lines following the printing of the last line of a report group.

13.18.36.1 General format

```
NEXT GROUP IS { integer-1                          }
              { { PLUS } integer-2                 }
              { { +    }                           }
              { NEXT PAGE [ WITH RESET ]           }
```

13.18.36.2 Syntax rules

1) Integer-1 specifies an absolute line number. Integer-2 specifies a relative vertical distance. Integer-1 and integer-2 shall not exceed the page limit, or 9999 if the report is not divided into pages. Integer-1 and integer-2 shall be unsigned. (See 13.18.38, PAGE clause.)

2) PLUS and + are synonyms.

3) If the report is not divided into pages, only the relative form of the clause may be specified.

4) The NEXT GROUP clause shall not be specified in a page heading or report footing.

5) The NEXT PAGE phrase shall not be specified in a page footing.

6) If the absolute form is used, the following checks apply:

   a) If the current report group is a report heading, integer-1 shall be greater than the minimum last line number of the report group and less than the FIRST DETAIL integer.

   b) If the current report group is a body group, integer-1 shall lie between the FIRST DETAIL integer and the FOOTING integer, inclusive.

   c) If the current report group is a page footing, integer-1 shall be greater than the minimum last line number of the report group.

7) If the relative form is used, the following checks apply:

   a) If the current report group is a report heading, the minimum last line number of the report group plus integer-2 shall be less than the FIRST DETAIL integer.

   b) If the current report group is a page footing, the minimum last line number of the report group plus integer-2 shall not exceed the page limit.

13.18.36.3 General rules

1) The NEXT GROUP clause has no effect when it is specified in a control footing that is at a level other than the highest level at which the control break is detected.

2) The NEXT GROUP clause modifies the value of the current report's LINE-COUNTER after the printing of the last line, if any, of the report group in whose description the clause appears. The effect of this clause depends on the type of report group whose description contains the clause, as covered in the following general rules.

3) If the report group is a report heading, the effect of the absolute and relative forms is to increase the line number on which the immediately next report group is printed, namely the first body group when the report

is not divided into pages, or the first page heading when the page heading consists only of relative lines. The effect on LINE-COUNTER in each case is as follows:

a) If the NEXT GROUP clause is absolute, LINE-COUNTER is set equal to integer-1.

b) If the NEXT GROUP clause is relative, integer-2 is added to LINE-COUNTER.

c) If NEXT GROUP NEXT PAGE is specified, the report heading is printed on the first page of the report as the only report group on that page and LINE-COUNTER is then set equal to zero.

4) If the report group is a body group:

a) If the NEXT GROUP clause is absolute, a check is made as to whether LINE-COUNTER is less than integer-1. If so, LINE-COUNTER is set equal to integer-1. Otherwise, integer-1 is placed in a save location and LINE-COUNTER is set equal to the FOOTING integer, causing a page advance to take place just before any other non-dummy body group is printed for the report. The NEXT GROUP clause has no further effect on the current body group and will have no effect at all if a TERMINATE is next executed for the report. But it will affect the location of the next non-dummy body group, at the time that it is printed, in the following way:

1. If the next body group begins with an absolute LINE clause without the NEXT PAGE phrase, the save location is moved to LINE-COUNTER and the page fit test is re-applied before the first line of the body group is printed.

2. If the next body group begins with an absolute LINE clause with the NEXT PAGE phrase, a page advance takes place, the save location is moved to LINE-COUNTER and a new page fit test and subsequent processing take place as for an identical report group without the NEXT PAGE phrase.

3. If the next body group contains only relative LINE clauses, its first line will be printed on the next line following the line number in the save location, unless this will result in some line of this body group being printed beyond its lower permitted limit. In the latter case, a second page advance takes place, resulting in a page devoid of body groups, and the next body group is printed on the following page with no reference to the save location.

b) If the NEXT GROUP clause is relative and if the sum of integer-2 and LINE-COUNTER is less than the FOOTING integer, integer-2 is added to LINE-COUNTER; otherwise the FOOTING integer is moved to LINE-COUNTER.

c) If NEXT GROUP NEXT PAGE is specified, the FOOTING integer is moved to LINE-COUNTER.

5) If the report group is a page footing, the NEXT GROUP clause affects any report footing defined in the current report using only relative LINE clauses, by changing LINE-COUNTER in the following ways:

a) If the NEXT GROUP clause is absolute, LINE-COUNTER is set equal to integer-1.

b) If the NEXT GROUP clause is relative, integer-2 is added to LINE-COUNTER.

6) If the WITH RESET phrase is present, the value of PAGE-COUNTER for the report is set to 1 (one) immediately after the page feed caused by the next page advance, chronologically between the printing of any page footing and the printing of any page heading. Whether the current group is a report heading or a body group, this phrase ensures that PAGE-COUNTER will be one, effective from the start of the next page (unless procedurally altered).

13.18.37 OCCURS clause

The OCCURS clause describes repeated data items, report items, and screen items and supplies information required for the application of subscripts.

13.18.37.1 General format

Format 1 (fixed-table):

```
OCCURS integer-2 TIMES
[ { ASCENDING  } KEY IS { data-name-2 } ... ] ... [ INDEXED BY { index-name-1 } ... ]
  { DESCENDING }
```

Format 2 (occurs-depending-table):

```
OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1
[ { ASCENDING  } KEY IS { data-name-2 } ... ] ... [ INDEXED BY { index-name-1 } ... ]
  { DESCENDING }
```

Format 3 (report-writer):

```
OCCURS [ integer-1 TO ] integer-2 TIMES [ DEPENDING ON data-name-1 ] [ STEP integer-3 ]
```

Format 4 (dynamic-capacity-table):

```
OCCURS DYNAMIC [ CAPACITY IN data-name-3 ] [FROM integer-4 ] [ TO integer-5 ] [ INITIALIZED ]
[ { ASCENDING  } KEY IS { data-name-2 } ... ] ... [ INDEXED BY { index-name-1 } ... ]
  { DESCENDING }
```

13.18.37.2 Syntax rules

ALL FORMATS

1) The OCCURS clause shall not be specified in a data description entry that:

   a) Has a level-number of 01, 66, 77, or 88, or

   b) Has an occurs-depending table subordinate to it.

2) Data-name-1 and data-name-2 may be qualified.

FORMATS 1, 2 AND 4

3) The first specification of data-name-2 shall be the name of either the entry containing the OCCURS clause or an entry subordinate to the entry containing the OCCURS clause. Subsequent specification of data-name-2 shall be subordinate to the entry containing the OCCURS clause.

4) If data-name-2 is subordinate to an alphanumeric group item, bit group item, national group item, or strongly-typed group item that is subordinate to the entry containing the OCCURS clause, that group item shall not contain an OCCURS clause.

5) Data-name-2 shall be specified without the subscripting normally required.

6) The data item identified by data-name-2 shall not contain an OCCURS clause except when data-name-2 is the subject of the entry.

7) Index-name-1 may be specified only in the following contexts:

   — as a subscript;
   — in the VARYING phrase of a PERFORM statement;
   — in the VARYING phrase of a SEARCH statement;
   — in the SET statement;
   — as an operand in a relation condition.

8) The KEY phrase shall not be specified for a data item of class boolean, object, or pointer.

9) Data-name-2 shall not reference a variable-length group.

FORMATS 1 AND 3

10) If the DEPENDING ON phrase is not specified, an OCCURS clause may be subordinate to a data description entry that contains another OCCURS clause as long as the number of subscripts required does not exceed seven.

FORMAT 1

11) An OCCURS clause specified in a screen description entry shall be a format 1 OCCURS clause without any optional phrases.

12) The maximum number of dimensions for a table described in a screen description entry is two.

13) If a screen description entry includes the OCCURS clause, then if it or any item subordinate to it has a description that includes the TO, FROM, or USING clause, that screen description entry shall be part of a table with the same number of dimensions and number of occurrences in each dimension as the identifier representing the receiving or sending operand. The identifier representing the receiving or sending operand shall not be subordinate to an OCCURS clause with the DEPENDING phrase.

14) If a screen description entry that includes the OCCURS clause also contains the COLUMN clause, then the COLUMN clause shall include the PLUS or MINUS phrase, unless the screen description entry also includes a LINE clause with a PLUS or MINUS phrase.

15) If a screen description entry that includes the OCCURS clause also contains the LINE clause, then the LINE clause shall include the PLUS or MINUS phrase, unless the screen description entry also includes a COLUMN clause with a PLUS or MINUS phrase.

FORMATS 2 AND 3

16) Integer-1 shall be greater than or equal to zero and integer-2 shall be greater than integer-1.

17) Data-name-1 shall describe an integer.

18) If the OCCURS clause is specified in an entry subordinate to one containing the GLOBAL clause, data-name-1, if specified, shall be a global name and shall reference a data item that is described in the same data division.

FORMAT 2

19) A format 2 OCCURS clause shall not be specified in any data item subordinate to a data item described with the CONSTANT RECORD clause.

20) The data item defined by data-name-1 shall not occupy a byte position within the range of the first byte position defined by the data description entry containing the OCCURS clause and the last byte position defined by the record description entry containing that OCCURS clause.

21) If the OCCURS clause is specified in a data description entry included in a record description entry containing the EXTERNAL clause, data-name-1 shall reference a data item possessing the external attribute that is described in the same data division.

22) The subject of the entry may be followed within that record description only by data description entries that are subordinate to it.

FORMAT 3

23) The DEPENDING phrase shall not be specified in any data item subordinate to a data item described with the CONSTANT RECORD clause.

24) The TO and DEPENDING phrases shall either be both absent or both present.

25) The STEP phrase shall be specified if the entry:

    a) contains an absolute LINE clause, or

    b) has an entry with an absolute LINE clause subordinate to it, or

    c) contains an absolute COLUMN clause, or

    d) is subordinate to an entry with a LINE clause and has an entry with an absolute COLUMN clause subordinate to it.

    In all other cases, the STEP phrase is optional.

26) The value of integer-3 shall be sufficient to prevent the overlapping of any line (in the case of vertical repetition) or column (in the case of horizontal repetition) of any two consecutive repetitions of the associated report item.

27) A report group description entry that contains an OCCURS clause with a DEPENDING phrase may be followed within that report group only by report group description entries that are subordinate to it.

FORMAT 4

28) Integer-4 shall be nonnegative and integer-5, if both are present, shall be greater than integer-4.

29) The implementor shall specify a maximum permissible value for integer-4 and integer-5. Their values shall not exceed this maximum value.

    NOTE This maximum should be more than enough to accommodate any capacity likely to be required by any error-free application running at the limit of its range.

30) Data-name-3 shall not be defined elsewhere in the source element. If qualifiers are required for uniqueness, it shall be treated as though implicitly defined at the same level as the entry containing the OCCURS clause.

31) Data-name-3 shall not be referenced as a receiving item, except as the operand of a variable-table format SET statement.

32) The dynamic-capacity-table format of the OCCURS clause shall not be specified in any data item described with the CONSTANT RECORD clause or any data item subordinate to a data item described with the CONSTANT RECORD clause.

13.18.37.3 General rules

FORMATS 1, 2 AND 4

1) Except for the OCCURS clause itself, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

2) The allocation and format of the index defined by index-name-1 are dependent on the implementor and the hardware. The implementor shall specify the rules for the range of values allowed in the index defined by index-name-1. This range shall include at least the value corresponding to the occurrence number (1 - integer-2) through and including the value corresponding to the occurrence number (2 * integer-2). An index may be modified only by a PERFORM VARYING statement, a SEARCH statement, and a SET statement. If the execution of one of these statements creates a value for the index that is outside the range of the values allowed by the implementor:

   a) the EC-RANGE-INDEX exception condition is set to exist, and

   b) the value of the index is undefined unless the value of the index is specified by the rules of that statement.

3) The KEY phrase indicates the order of key data items used during execution of a SEARCH statement with the ALL phrase specified, or during the execution of a SORT statement that references a table. If more than one data-name-2 is specified, they are specified in descending order of significance. The data associated with data-name-2 shall be ordered when a SEARCH statement with the ALL phrase is executed if data-name-2 is specified in one of the conditions in the WHEN phrase. At the time of the execution of such a SEARCH statement, the contents of the data items referenced by data-name-2 shall be in ascending order if the ASCENDING phrase is specified or descending order if the DESCENDING phrase is specified. The associated collating sequence for the order is determined by the rules for comparison of operands that apply to the condition specified in the WHEN phrase of the SEARCH statement.

FORMAT 1

4) The value of integer-2 represents the fixed number of occurrences of the subject of the entry.

5) During a DISPLAY screen or an ACCEPT screen statement that references a screen item whose description includes the OCCURS clause and whose description or whose subordinate's description includes a FROM, TO, or USING clause, the data values for corresponding table elements are moved from the data table element to the screen table element or from the screen table element to the data table element.

6) If the description of a screen item includes the OCCURS clause, the positioning within the screen record of each occurrence of that screen item is as follows:

   a) If the description of that screen item contains a COLUMN clause, each occurrence behaves as though it had the same COLUMN clause specified.

   b) If that screen item is a group item with a subordinate screen item whose description contains a COLUMN clause with the PLUS or MINUS phrase and that group screen item is subordinate to a screen item whose description contains a LINE clause, each occurrence behaves as though it had the same subordinate entries with the same COLUMN clause specified.

   c) If the description of that screen item contains a LINE clause with the PLUS or MINUS phrase, each occurrence behaves as though it had the same LINE clause specified.

d) If that screen item is a group item with a subordinate screen item whose description contains a LINE clause with the PLUS or MINUS phrase, each occurrence behaves as though it had the same subordinate entries with the same LINE clause specified.

FORMAT 2

7) The value of the data item referenced by data-name-1 represents the current number of occurrences of the subject of the entry.

   The subject of this entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable.

   At the time the subject of entry is referenced or any data item subordinate or superordinate to the subject of entry is referenced, the value of the data item referenced by data-name-1 shall fall within the bounds from integer-1 through integer-2. If the value of the data item does not fall within the specified bounds, the EC-BOUND-ODO exception condition is set to exist. The content of a data item whose occurrence number exceeds the value of the data item referenced by data-name-1 is undefined.

8) An alphanumeric group item, bit group item, national group item, or strongly-typed group item that has an entry subordinate to it that specifies the DEPENDING phrase of the OCCURS clause is an occurs-depending group item. When an occurs-depending group item is referenced, the part of the table area used in the operation is determined as follows:

   a) If the data item referenced by data-name-1 is outside the group, only that part of the table area that is specified by the value of the data item referenced by data-name-1 at the start of the operation will be used. If there are no elementary data items defined between the data description entry of the group data item and the definition of the table specified by format 2 and the value of the data item referenced by data-name-1 at the start of the operation is zero, the group data item is a zero-length item.

   b) If the data item referenced by data-name-1 is included in the same group and the group data item is referenced as a sending operand, only that part of the table area that is specified by the value of the data item referenced by data-name-1 at the start of the operation will be used in the operation. If the group is a receiving operand, the maximum length of the group will be used.

9) If format 2 is specified in a record description entry and the associated file description or sort-merge description entry contains the VARYING phrase of the RECORD clause, the records are variable length. If the DEPENDING ON phrase of the RECORD clause is not specified, the content of the data item referenced by data-name-1 of the OCCURS clause shall be set to the number of occurrences to be written before the execution of any RELEASE, REWRITE, or WRITE statement referencing that record description entry.

FORMAT 3

10) If the OCCURS clause is written without any of the optional phrases, it causes the entry to define integer-2 distinct report items. The effect of the OCCURS clause on each repetition depends on the position of the entry containing the clause in the report group definition, as follows:

    a) If the entry also contains a relative COLUMN clause, each repetition behaves as though it had the same relative COLUMN clause.

    b) If the entry is a group entry having subordinate entries with relative COLUMN clauses, and being itself subordinate to an entry with a LINE clause, each repetition behaves as though it had the same subordinate entries with the same relative COLUMN clauses.

    c) If the entry also contains a relative LINE clause, each repetition behaves as though it had the same relative LINE clause.

d) If the entry is a group entry having subordinate entries with relative LINE clauses, each repetition behaves as though it had the same subordinate entries with the same relative LINE clauses.

11) Any PICTURE, USAGE, SIGN, VALUE, JUSTIFIED, BLANK WHEN ZERO, or GROUP INDICATE clauses have the same effect on each repetition as they would on a single data item without the OCCURS clause. This applies also to any SOURCE, SUM, or PRESENT WHEN clauses if no VARYING clause is present. If a VARYING clause is present, the action of these clauses may vary from one repetition to another. (See 13.18.63, VARYING clause.)

12) The STEP phrase, if specified, defines the vertical or horizontal interval between successive occurrences of the associated report item after the first occurrence, as follows:

a) If the entry contains a COLUMN clause, each successive occurrence is printed at a horizontal distance integer-3 columns to the right of the preceding occurrence.

b) If the entry is a group entry having subordinate entries with COLUMN clauses and being itself subordinate to an entry with a LINE clause, printable items in each successive occurrence are positioned integer-5 columns to the right of the column they occupy in the preceding occurrence.

c) If the entry contains a LINE clause, each successive occurrence is positioned integer-3 lines vertically beneath the preceding occurrence.

d) If the entry is a group entry having subordinate entries with LINE clauses, report lines in successive occurrences are positioned integer-3 lines vertically beneath the line they occupy in the preceding occurrence.

If no STEP phrase is specified, the vertical or horizontal interval between successive occurrences is defined by the relative LINE or COLUMN numbers, respectively, specified in the corresponding report section entries.

13) If the DEPENDING phrase is specified, the value of data-name-1 is evaluated just before the processing for the first LINE clause of the report group. If the value of data-name-1 is not in the range integer-1 to (integer-2 - 1), the report group is processed as though the OCCURS clause had been written without the TO and DEPENDING phrases. If the value of data-name-1 is in the range integer-1 to (integer-2 - 1), the OCCURS clause has the same effect as an OCCURS clause with no TO or DEPENDING phrases and with an integer-2 equal to the current value of data-name-1. This same principle is used in the computing of the trial sum used in performing the page fit test. (See 13.18.34, LINE clause, general rule 4c.)

FORMAT 4

14) Format 4 defines a dynamic-capacity table. Other rules and restrictions concerning dynamic-capacity tables are specified in 8.5.1.6.3, Dynamic-capacity tables.

15) Data-name-3 defines a numeric data item that contains the current capacity of the associated table. Data-name-3 shall not be referenced as a receiving operand.

16) Integer-4 is the minimum capacity of the table. If integer-4 is absent, a value of zero is assumed for it.

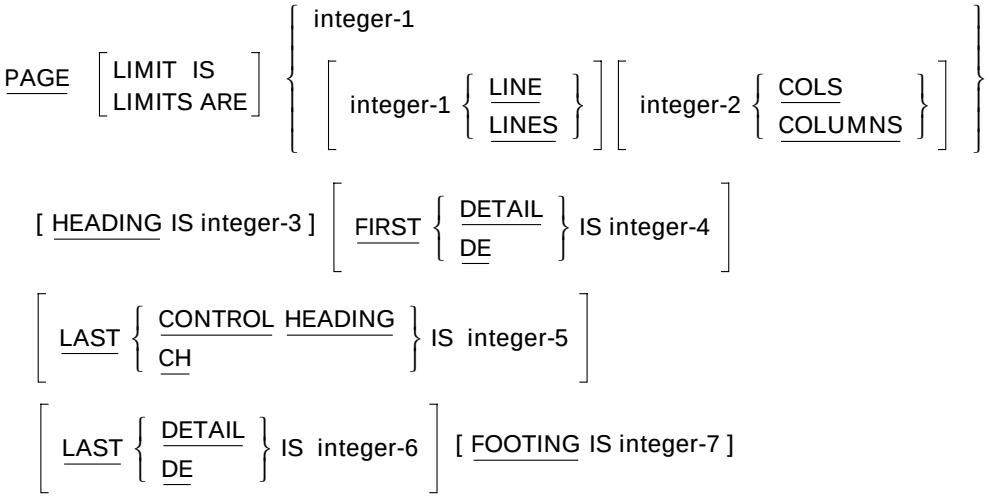
17) Integer-5 is the expected capacity of the table.

18) If INITIALIZED is specified, any unreferenced locations of each new entry added to the table are implicitly initialized, as defined in 8.5.1.6.3.4, Implicit initialization.

13.18.38 PAGE clause

The PAGE clause defines the maximum length and width of a page of a report and the vertical subdivisions within which its report groups shall be printed.

13.18.38.1 General format



13.18.38.2 Syntax rules

1) FIRST DE is synonymous with FIRST DETAIL, LAST CH is synonymous with LAST CONTROL HEADING, and LAST DE is synonymous with LAST DETAIL.

2) Either integer-1 or integer-2 or both shall be specified.

3) The HEADING, FIRST DETAIL, LAST CONTROL HEADING, LAST DETAIL, or FOOTING phrase may be specified only if integer-1 is specified.

4) The HEADING, FIRST DETAIL, LAST CONTROL HEADING, LAST DETAIL, and FOOTING phrases may be written in any order.

5) Integer-1 shall not exceed 9999.

6) Integer-3, integer-4, integer-5, integer-6, integer-7, and integer-1 shall be greater than zero. Wherever specified, they shall be in ascending order, with equality allowed.

13.18.38.3 General rules

1) The printed page is regarded as a vertical arrangement of horizontal lines, each line being capable of displaying the content of one report line. On each page these lines are numbered from 1 (representing the first line on the page) in steps of 1. The PAGE clause and its phrases subdivide the printed page into vertical regions. Depending on its type, each report group shall be confined within one or more of these regions.

2) The integers specified in the PAGE clause establish the vertical regions of the report page as follows.

   a) Integer-1 is the page limit. It defines the last line position on each page of the report. No report line will appear below this position on any page. If integer-1 is not specified, the report consists of a single page of indefinite length.

b) Integer-2 is the page width. It defines the maximum number of print columns that may be accommodated in any line of the report.

c) Integer-3 is the HEADING integer. It defines the first line position on which a report heading or page heading may be printed. No report line will appear higher than this position on the page.

d) Integer-4 is the FIRST DETAIL integer. It defines the first line position on which any line of a body group may be printed. Any report heading (when not on a page by itself) or page heading shall be defined so that it terminates before this line.

e) Integer-5 is the LAST CONTROL HEADING integer. It defines the last line position on which any line of a control heading may be printed.

f) Integer-6 is the LAST DETAIL integer. It defines the last line position on which any line of a detail may be printed.

g) Integer-7 is the FOOTING integer. It defines the last line position on which any line of a control footing may be printed. Any page footing or report footing (when not on a page by itself) shall be defined so that it begins after this line.

3) If integer-1 is specified and any of the following phrases is omitted, a default value for the associated integer is supplied as follows:

a) If HEADING is omitted, integer-3 will be 1.

b) If FIRST DETAIL is omitted, integer-4 will be equal to integer-3.

c) If LAST CONTROL HEADING is omitted, integer-5 will be equal to integer-6, if LAST DETAIL is specified, or equal to integer-7, if FOOTING is specified, or otherwise equal to the page limit given by integer-1.

d) If LAST DETAIL is omitted, integer-6 will be equal to integer-7, if FOOTING is specified, or otherwise equal to the page limit given by integer-1.

e) If FOOTING is omitted, integer-7 will be equal to integer-6, if LAST DETAIL is specified, or otherwise equal to the page limit given by integer-1.

4) A report heading or report footing on a page by itself may occupy any region of the page from HEADING integer-3 through the page limit integer-1 or, if not on a page by itself, is constrained in the same way as a page heading or page footing respectively.

5) If integer-2 is omitted, a value of 999 is assumed for the page width.

13.18.39 PICTURE clause

The PICTURE clause describes the general characteristics, editing requirements, and format validation profile of an elementary item.

13.18.39.1 General format

Format 1 (basic):

```
{ PICTURE } IS character-string-1
{ PIC     }
```

Format 2 (locale):

```
{ PICTURE } IS character-string-1 LOCALE [ IS locale-name-1 ] SIZE IS integer-1
{ PIC     }
```

13.18.39.2 Syntax rules

ALL FORMATS

1) The PICTURE clause may be specified only at the elementary level.

2) Character-string-1 shall consist of an allowable combination of characters used as picture symbols.

   The allowable combinations of symbols for a PICTURE clause are specified in 13.18.39.5, Precedence rules.

   NOTE The currency symbol may be selected from the set of allowable characters from the computer's compile-time coded character set. All picture symbols other than the currency symbol are from the COBOL character repertoire.

3) The equivalence between uppercase and lowercase letters appearing as occurrences of a currency symbol in character-string-1 is as specified in 12.3.6, SPECIAL-NAMES paragraph, syntax rule 20. The equivalence between uppercase and lowercase letters appearing as picture symbols other than currency symbols in character-string-1 is as specified in 8.1.2, COBOL character repertoire, syntax rule 3.

4) The maximum number of characters allowed in character-string-1 is 50.

5) PIC is an abbreviation for PICTURE.

6) An unsigned nonzero integer that is enclosed in parentheses indicates the number of consecutive occurrences of the symbol that immediately precedes the left parenthesis. The integer may be specified by a constant-name, in which case the length of the integer, not the length of the constant-name, is counted toward the maximum number of characters in character-string-1.

7) If the symbol ',' or the symbol '.' is the last symbol of character-string-1, the PICTURE clause shall be the last clause of the data description entry and shall be followed immediately (without an intervening separator space) by the separator period.

FORMAT 1

8) Character-string-1 shall contain

   — at least one of the symbols from the set 'A', 'N', 'X', 'Z', '1', '9', '*'; or
   — at least two occurrences of one of the symbols from the set '+', '–', the currency symbol.

9) Each of the symbols from the set 'CR', 'DB', 'E', 'S', 'V', '.' may appear only once in character-string-1.

NOTE The symbols 'CR' and 'DB', although consisting of two characters, are each considered to be a symbol by itself.

10) When the DECIMAL-POINT IS COMMA clause is specified, the symbol comma is the decimal separator and the symbol period is the grouping separator. The rules for the symbol period apply to the symbol comma, and the rules for the symbol comma apply to the symbol period.

11) For data items of category numeric, the number of digit positions described by character-string-1 shall range from 1 through 31.

12) For floating-point data items of category numeric-edited, the number of digit positions in the significand portion of character-string-1 shall range from 1 through 34.

13) The symbol 'P' may appear only as a continuous string of 'P's in the leftmost or rightmost digit positions in character-string-1.

14) The symbol 'P' and the symbol '.' are mutually exclusive in character-string-1.

15) The symbol 'S', if present, shall be the first symbol in character-string-1.

16) When the symbol 'V' and one or more symbols 'P' are used in character-string-1, the symbol 'V' shall either immediately precede the first symbol 'P' or immediately follow the last symbol 'P'.

17) The symbol 'V' and the symbol '.' are mutually exclusive in character-string-1.

18) The symbol 'Z' and the symbol '*' are mutually exclusive in character-string-1.

19) Neither the symbol 'S' nor the symbol '*' shall be specified in character-string-1 when the BLANK WHEN ZERO clause is specified for the subject of the entry.

20) The editing sign control symbols '+', '-', 'CR', and 'DB' are mutually exclusive in character-string-1 with the exception of a numeric-edited data item for a floating-point edited result as described in general rule 12b.

NOTE For a floating-point edited result, the significand part of the character-string may contain a '-' symbol and the exponent part always contains a '+' symbol.

21) For fixed insertion, only one currency symbol and only one editing sign control symbol may be used in character-string-1.

22) The symbol '+' or the symbol '-', when used, shall be either the leftmost or the rightmost symbol in character-string-1.

23) The symbol 'CR' or the symbol 'DB', when used, shall be the rightmost symbol in character-string-1.

24) The currency symbol, when used, shall be either the leftmost symbol in character-string-1, optionally preceded by one of the symbols '+' or '-', or the rightmost symbol in character-string-1, optionally followed by one of the symbols '+', '-', 'CR', or 'DB'.

NOTE This means that the following are valid picture character-strings:
'$999+'
'+999$'
'+$999'
'999$+'
'+$$99'
'$$99+'.

25) No more than one of the following may be specified in character-string-1:

— - a string of two or more symbols '+';
— - a string of two or more symbols '-';
— - a string of two or more currency symbols;
— - a string of one or more symbols '*';
— - a string of one or more symbols 'Z'.

NOTE This means, for example, that the picture character-string +$$$ is valid, but that the picture character-string +++$$$ is invalid.

26) For floating insertion, when the currency symbol is used as the floating insertion symbol, all occurrences of the currency symbol shall be equivalent.

27) For floating insertion, at least one insertion symbol shall be specified to the left of the decimal point position.

28) The symbols 'A' and 'X' shall not be specified in character-string-1 when a USAGE NATIONAL clause is specified for the subject of the entry.

FORMAT 2

29) A format 2 PICTURE clause shall not be specified in a data item described with the CONSTANT RECORD clause, or in any data item subordinate to a data item described with the CONSTANT RECORD clause.

30) Character-string-1 shall contain at least one of the symbols 'Z' or '9'.

31) Each of the symbols from the set '+', '.', the currency symbol may appear only once in character-string-1.

32) The number of digit positions described by character-string-1 shall range from 1 through 31.

33) The currency symbol and the symbol '+' may be specified only to the left of the decimal point position.

34) Locale-name-1 shall be specified in the LOCALE clause in the SPECIAL-NAMES paragraph.

13.18.39.3 General rules

ALL FORMATS

1) When the usage of the subject of the entry is national, each symbol representing a character position defines a national character position. When the usage of the subject of the entry is display, each symbol representing a character position defines an alphanumeric character position.

2) The value of insertion and replacement characters in a resultant edited item is the value of those characters in the computer's runtime coded character set. When the usage of the item being edited is national, the value is the national character representation; otherwise, the value is the alphanumeric character representation.

FORMAT 1

3) A PICTURE clause defines the subject of the entry to fall into one of the following categories of data:

— alphabetic
— alphanumeric
— alphanumeric-edited
— boolean
— national
— national-edited
— numeric
— numeric-edited

A BLANK WHEN ZERO clause specified for the subject of the entry defines the item as numeric-edited.

4) The size in boolean positions or character positions of an elementary data item that has been defined with a PICTURE clause is determined by the number of symbols in character-string-1 that represent either boolean positions or character positions.

5) To define an item as alphabetic, character-string-1 shall contain only one or more occurrences of the symbol 'A'.

6) To define an item as alphanumeric, character-string-1 shall contain a combination of symbols from the set 'A', 'X', and '9', that includes

   — at least one symbol 'X', or
   — at least two different symbols from this set.

7) To define an item as alphanumeric-edited, character-string-1 shall include

   — at least one symbol 'A' or one symbol 'X', and
   — at least one of the symbols from the set 'B', '0', '/'.

8) To define an item as boolean, character-string-1 shall contain only one or more occurrences of the symbol '1'.

9) To define an item as national, character-string-1 shall contain only one or more occurrences of the symbol 'N'.

10) To define an item as national-edited, character-string-1 shall include

   — at least one symbol 'N', and
   — at least one of the symbols from the set 'B', '0', '/'.

11) To define an item as numeric, character-string-1

   — shall include at least one symbol '9', and
   — may contain a combination of symbols from the set 'P', 'S', and 'V'.

12) To define an item as numeric-edited, one of the following options shall be specified:

   a) To produce a fixed-point edited result, character-string-1 shall include:

      — at least one symbol 'Z'; or

      — at least one symbol '*'; or

      — at least two identical symbols from the set '+', '-', currency symbol; or

      — at least one symbol '9' and at least one of the symbols from the set 'B', 'CR', 'DB', '0', '/', ',', '.', '+', '-', the currency symbol.

   b) To produce a floating-point edited result, characters-string-1 shall consist of two parts, separated without any spaces, by the symbol 'E'. The first part represents the significand; the second part represents the exponent.

      The significand shall be a valid character-string for either a numeric item or a numeric-edited item for a fixed-point result. Neither floating insertion editing nor zero suppression with replacement shall be specified for the significand.

      The exponent shall be ‘+9’, ‘+99’, ‘+999’, ‘+9999’, or ‘+9(n)’ where n = 1, 2, 3, or 4.

13) The meaning of the symbols used in character-string-1 are as follows:

A Each symbol 'A' represents a character position that shall contain any character from the computer's alphanumeric character set. Each symbol 'A' is counted in the size of the item.

The characters represented may be graphic characters or non-graphic characters.

B Each symbol 'B' represents a character position into which the character space will be inserted during editing. Each symbol 'B' is counted in the size of the item.

E The symbol 'E' represents a character position into which the character 'E' will be inserted during editing. The symbol 'E' is counted in the size of the item.

The symbol 'E' is used to separate the significand and the exponent of a floating-point numeric-edited item.

N Each symbol 'N' represents a national character position that shall contain a character from the computer's national character set. Each symbol 'N' is counted in the size of the item.

P The symbol 'P' specifies the location of an assumed decimal point when that point is not within the number that appears in the data item. The symbol 'P' is not counted in the size of the item, but each symbol 'P' is counted in the maximum number of digit positions.

The symbol 'P' implies an assumed decimal point that is either

- to the left of the string of 'P's if they indicate the leftmost digit positions in character-string-1; or
- to the right of the string of 'P's if they indicate the rightmost digit positions in character-string-1.

In certain operations that reference a data item whose picture character-string contains the symbol 'P', the algebraic value of the data item is used rather than the actual value of the data item. This algebraic value assumes the decimal point in the prescribed position and zero in place of each digit position specified by the symbol 'P'. The size of the value is the number of digit positions represented by the picture character-string.

The operations that use the algebraic value are the following:

a) Any operation requiring a numeric sending operand.

b) An elementary move operation where the sending operand is numeric and its picture character-string contains one or more symbols 'P'.

c) A move operation where the sending operand is numeric-edited, its picture character-string contains one or more symbols 'P', and the receiving operand is numeric or numeric-edited.

d) A comparison operation where both operands are numeric.

In all other operations, the digit positions specified with the symbol 'P' are ignored and are not counted in the size of the operand.

S The symbol 'S' indicates the presence, but neither the representation nor, necessarily, the position of an operational sign. For usages display and national, the symbol 'S' is counted in the size of the item only when the subject of the entry is described with a SIGN clause with the SEPARATE phrase. For other numeric usages, the effect of the 'S' is described in the rules for the USAGE clause.

V The symbol 'V' indicates the position of the assumed decimal point for alignment purposes. The symbol 'V' is not counted in the size of the item.

When the assumed decimal point position is to the right of the rightmost digit position, the symbol 'V' is redundant.

X Each symbol 'X' represents a character position that shall contain any character from the computer's alphanumeric character set. Each symbol 'X' is counted in the size of the item.

The characters represented may be graphic characters or non-graphic characters.

Z Each symbol 'Z' represents a leading numeric position that during editing will contain a numeric character in the range 0 through 9, or a character space when the content of that position is a leading zero. Each symbol 'Z' is counted in the size of the item.

0 Each symbol '0' (zero) represents a character position into which the character zero will be inserted during editing. Each symbol '0' is counted in the size of the item.

1 Each symbol '1' represents a boolean position that shall contain a boolean character. Each symbol '1' is counted in the size of the item.

9 Each symbol '9' represents a decimal digit position of the value of the item. For usages display and national, each '9' represents a character position that shall contain a numeric character in the range 0 through 9 with the possible inclusion of an operational sign; each '9' is counted in the size of the item. For other numeric usages, the effect of the number of '9's is described in the rules for the USAGE clause.

/ Each symbol '/' (slant) represents a character position into which the character slant will be inserted during editing. Each symbol '/' is counted in the size of the item.

, Each symbol ',' (comma) represents a character position into which the character comma will be inserted during editing. Each symbol ',' is counted in the size of the item.

. The symbol '.' (period) represents a character position into which the character period will be inserted during editing. The symbol '.' is counted in the size of the item.

In addition, the symbol '.' indicates the decimal point position for alignment purposes.

+ —CR DB These symbols represent the character position(s) into which the editing sign control character(s) is (are) placed during editing. Each character used in the symbol is counted in the size of the item.

* Each symbol '*' represents a leading numeric position that during editing will contain a numeric character in the range 0 through 9, or a character asterisk when the content of that position is a leading zero. Each symbol '*' is counted in the size of the item.

cs A currency symbol represents character positions into which the currency string will be placed during editing. A currency symbol is represented in character-string-1 either by the currency sign or by the currency symbol specified in a CURRENCY SIGN clause in the SPECIAL-NAMES paragraph.

The first occurrence of the currency symbol adds the number of characters in the currency string to the size of the item. Each subsequent occurrence of the currency symbol adds one to the size of the item.

14) The PICTURE clause takes effect during the format validation stage of the execution of a VALIDATE statement that refers directly or indirectly to the subject of the entry.

If the usage of the subject of the entry is not display or national, character-string-1 is used in combination with the USAGE clause to check, where applicable, that the contents are compatible with character-string-1. The rules for compatibility are implementor-defined.

If the usage of the subject of the entry is binary, computational or packed-decimal, and the range of values permitted by character-string-1 is less than that used by the hardware for the representation of the item, the unused portion of the hardware allocation is checked to contain only the value zero.

If the usage is display or national, the effects of each symbol in character-string-1 are explained as follows:

A Each symbol 'A' represents a character position that will be checked to contain an alphabetic character. The classification of characters as alphabetic is determined in accordance with the rules for the ALPHABETIC class condition as specified in 8.8.4.1.3, Class condition.

B Each symbol 'B' represents a character position that will be checked to contain

- the character space if the symbol 'B' is neither part of floating insertion editing nor of zero suppression with replacement editing;
- otherwise, the character space or, if no significant numeric character appears to its left, the corresponding floating insertion character or replacement character respectively.

E The symbol 'E' represents a character position that will be checked to contain the character ‘E’.

N Each symbol 'N' represents a character position that will be checked to contain a character from the computer's national character set.

S The symbol 'S' specifies that the data item will be checked for the presence of an operational sign.

- If the subject of the entry is not specified with a SIGN clause, the position and the mode of representation of this operational sign are checked according to the specifications of the implementor;
- If the subject of the entry is specified with a SIGN clause without the SEPARATE phrase, then either the first or the last digit position and the mode of representation of the operational sign are checked according to the specifications of the implementor;
- If the subject of the entry is specified with a SIGN clause with the SEPARATE phrase, then the corresponding position of the data item is checked to contain the character '+' or the character '-'.

Z Each symbol 'Z' represents a character position that will be checked to contain:

- the space character, if the symbol 'Z' is specified to the left of the decimal point position and the content of the character position is a leading zero, or
- the space character, if the symbol 'Z' is specified at the rightmost digit position after the decimal point position and the content of the data item is zero, or
- a numeric character in the range 0 through 9 otherwise.

0 Each symbol '0' (zero) represents a character position that will be checked to contain

- the character zero if the symbol '0' is neither part of floating insertion editing nor of zero suppression with replacement editing;
- otherwise, the character zero or, if no significant numeric character appears to its left, the corresponding floating insertion character or replacement character respectively.

1 Each symbol '1' represents a boolean position that will be checked to contain a boolean character.

9 Each symbol '9' represents a character position that will be checked to contain a numeric character in the range 0 through 9 with the possible inclusion of an operational sign.

/ Each symbol '/' (slant) represents a character position that will be checked to contain

- the character slant if the symbol '/' is neither part of floating insertion editing nor of zero suppression with replacement editing;
- otherwise, the character slant or, if no significant numeric character appears to its left, the corresponding floating insertion character or replacement character respectively.

, Each symbol ',' (comma) represents a character position that will be checked to contain

- the character comma if the symbol ',' is neither part of floating insertion editing nor of zero suppression with replacement editing;
- otherwise, the character comma or, if no significant numeric character appears to its left, the corresponding floating insertion character or replacement character respectively.

. The symbol '.' (period) represents a character position that will be checked to contain the character period.

+ – CR DB These symbols represent character positions that will be checked to contain a valid sign, according to the following:

- the character position corresponding to the symbol '+' or '-' when such a symbol is used as a fixed insertion symbol, will be checked to contain a valid sign character as defined in table 9, Results of fixed insertion editing;
- the character positions corresponding to the symbols '+' or '-' when such symbols are used as floating insertion symbols, will all be checked to contain valid characters, including a valid sign character, as specified in 13.18.39.4, Editing rules, rule 6;
- the character positions corresponding to the symbols 'CR' and 'DB' will be checked to contain valid sign characters as defined in table 9, Results of fixed insertion editing.

* Each symbol '*' represents a character position that will be checked to contain:

- the character asterisk, if the symbol '*' is specified to the left of the decimal point position and the content of the character position is a leading zero, or
- the character asterisk, if the symbol '*' is specified at the rightmost digit position after the decimal point position and the content of the data item is zero, or
- a numeric character in the range 0 through 9 otherwise.

cs The character position(s) corresponding to the currency symbol will be checked as follows:

- when used as a fixed insertion symbol, the character position(s) will be checked to contain the currency string,

- when used as floating insertion symbols, the character position(s) will be checked to contain valid characters, including a valid currency string, as specified in 13.18.39.4, Editing rules, rule 6.

All other symbols have no effect on format validation.

FORMAT 2

15) Format 2 of the PICTURE clause defines the item to be fixed-point numeric-edited.

16) The number of character positions in the item is specified by integer-1.

17) The meaning of the symbols used in character-string-1 are as follows:

Z Each symbol 'Z' represents a leading numeric position that during editing will contain a numeric character in the range 0 through 9, or a character space when the content of that position is a leading zero.

9 Each symbol '9' represents a digit position that shall contain a numeric character in the range 0 through 9.

. The symbol '.' (period) represents a character position into which the decimal separator taken from the locale will be inserted during editing.

In addition, the symbol '.' indicates the decimal point position for alignment purposes.

+ The symbol '+' indicates that the item is to be signed in accordance with the specifications in the locale. If the symbol '+' is not specified, the item will be unsigned.

cs The currency symbol indicates that the item is to include a currency string in accordance with the specifications in the locale.

18) The PICTURE clause takes effect during the format validation stage of the execution of a VALIDATE statement that refers directly or indirectly to the subject of the entry. The validation is carried out in accordance with the character classification and monetary specification in the locale.

13.18.39.4 Editing rules

FORMAT 1

1) There are two methods of performing editing: either insertion or suppression with replacement.

There are four types of insertion editing:

— simple insertion
— special insertion
— fixed insertion
— floating insertion

There are two types of suppression with replacement:

— zero suppression with replacement with spaces
— zero suppression with replacement with asterisks

2) The type of editing that may be performed upon an item is dependent upon the category to which the item belongs. Table 8, Category and type of editing, specifies which type of editing may be performed upon a given category:

Table 8 — Category and type of editing

Category	Type of editing
Alphabetic	None
Alphanumeric	None
Boolean	None
National	None
Numeric	None
Alphanumeric-edited	Simple insertion
National-edited	Simple insertion
Numeric-edited (fixed-point edited result)	All
Numeric-edited (floating-point edited result)	Simple insertion, special insertion, and fixed insertion for the significand part None for the exponent part

3) Simple insertion editing

The symbols 'B', '0', '/', and ',' are used as the simple insertion editing symbols.

Simple insertion editing results in the insertion character occupying the same character position in the edited item as the associated symbol occupies in character-string-1.

4) Special insertion editing

The symbol '.' is used as the special insertion editing symbol.

Special insertion editing results in the period character occupying the same character position in the edited item as the symbol '.' occupies in character-string-1.

5) Fixed insertion editing

The currency symbol and the editing sign control symbols '+', '-', 'CR' and 'DB' are used as the fixed insertion editing symbols.

Fixed insertion editing results in the insertion character(s) occupying the same character position(s) in the edited item as the associated symbol occupies in character-string-1.

Table 9, Results of fixed insertion editing, shows the character(s) produced by an editing sign control symbol, depending on the value of the data item.

Table 9 — Results of fixed insertion editing

Editing symbol	Result	
	Positive or zero value	Negative value
+	+	–
–	space	–
CR	2 spaces	CR
DB	2 spaces	DB

The uppercase letters CR and DB are the insertion characters for the insertion symbols 'CR' and 'DB'.

6) Floating insertion editing

The currency symbol and the editing sign control symbols '+' and '-' are used as the floating insertion symbols.

Floating insertion editing is indicated by specifying a string of at least two identical floating insertion editing symbols. Any of the simple insertion editing symbols embedded in this string or to the immediate right of this string are part of the string.

The leftmost symbol of the insertion string represents the leftmost limit of the floating characters in the data item. The rightmost symbol of the insertion string represents the rightmost limit of the floating characters in the data item.

The second floating symbol represents the leftmost limit of the numeric data that may be stored in the item. During editing, nonzero numeric characters may replace all the insertion symbols at or to the right of this limit.

If truncation occurs, the value of the data that is used for editing is the value after truncation as specified in 14.6.8, Alignment of data within data items.

NOTE To avoid unwanted truncation of data, the minimum size of character-string-1 should be the number of characters in the sending operand, plus the number of non-floating insertion symbols to be inserted in the item, plus one.

There are two ways of representing floating insertion editing:

a) one way is to represent any or all of the leading numeric character positions to the left of the decimal point position by the same insertion symbol. The result is that a single occurrence of the replacement character(s) is (are) placed into the character position(s) immediately preceding whichever of the following is encountered first:

— the first nonzero numeric character in the item
— the first character position for which no floating insertion editing is specified

— the decimal point position.

Any character positions preceding this (these) insertion character(s) will contain the space character.

b) the second way is to represent all of the numeric character positions by the same insertion symbol. The result depends upon the value of the data to be stored. If the value is not zero, the result is the same as if the floating insertion editing were defined only to the left of the decimal point position. If the value is zero, all character positions will contain the space character.

Table 10, Results of floating insertion editing, shows the character produced by the floating editing sign control symbols '+' and '-', depending on the value of the data item.

Table 10 — Results of floating insertion editing

Editing symbol in picture character-string	Result	
	Data item positive or zero	Data item negative
+	+	–
–	space	–

7) Zero suppression with replacement editing

The symbols 'Z' and '*' are used as the symbols for zero suppression with replacement. If the symbol 'Z' is used, the replacement character is the character space; if the symbol '*' is used, the replacement character is the character asterisk.

Zero suppression with replacement is indicated by specifying a string of one or more identical zero-suppression characters. Any of the simple insertion editing symbols embedded in this string or to the immediate right of this string are part of the string.

There are two ways of representing zero suppression with replacement:

a) one way is to represent any or all of the leading numeric character positions to the left of the decimal point position by the zero-suppression symbol. The result is that the corresponding replacement character is placed into any character position immediately preceding whichever of the following is encountered first:

— the first nonzero numeric character in the item
— the first character position for which no zero suppression with replacement is specified
— the decimal point position.

b) the second way is to represent all of the numeric character positions by the zero-suppression symbol. The result depends upon the value of the data to be stored. If the value is not zero, the result is the same as if the zero suppression with replacement were defined only to the left of the decimal point position. If the value is zero and the zero-suppression symbol is the symbol 'Z', all character positions of the item will contain the character space. If the value is zero and the zero-suppression symbol is the symbol '*', all character positions of the item will contain the character asterisk, but the decimal separator, when specified, will appear in the item.

8) Zero value of a floating-point edited item

If the value to be edited into a floating-point edited item is zero, then after editing all digit positions of the significand and all digit positions of the exponent shall be zero; the sign of the significand, if present, shall be positive; and the sign of the exponent shall be positive.

FORMAT 2

9) The locale category LC_MONETARY is used for locale editing. The position, length, and character(s) used for the currency symbol are determined from that locale category.

10) A BLANK WHEN ZERO clause takes precedence over locale editing.

11) When locale-name-1 is specified, the locale used in editing and de-editing the item is the one associated with that name by the LOCALE clause in the SPECIAL-NAMES paragraph; otherwise, the current locale is used.

    NOTE Switching locales between editing and de-editing of a given data item may result in unpredictable behavior. The programmer is responsible for ensuring that the locale used for de-editing is the same as the one used for editing.

12) The decimal separator, the grouping separator, and the number of digits in a group are determined from the locale category LC_MONETARY.

13) If the symbol '+' is specified, the manner of representing positive and negative numbers is determined from the locale.

14) During editing, the data is aligned on the decimal point position with zero fill or truncation on either end, within a hypothetical data item, with grouping and separators in accordance with the locale specifications.

    The hypothetical data item is then moved to the data item being edited. The following rules apply:

    a) If the size of the data item is larger than the size of the hypothetical data item, the data is right justified, with space fill for the leftmost character positions.

    b) If the size of the data item is smaller than the size of the hypothetical data item, the leftmost character positions of the hypothetical data item are truncated. If any truncated character is neither a zero nor a space caused by a suppressed zero, the EC-LOCALE-SIZE exception condition is set to exist.

15) Zero suppression with replacement editing

    The symbol 'Z' is used as the symbol for zero suppression with replacement. The replacement character is the character space.

    Zero suppression with replacement is indicated by specifying a string of one or more symbols 'Z'.

    There are two ways of representing zero suppression with replacement:

    a) one way is to represent any or all of the leading numeric character positions to the left of the decimal point position by the symbol 'Z'. The result is that the space character is placed into any character position immediately preceding whichever of the following is encountered first:

       — the first nonzero numeric character in the item
       — the first character position for which no zero suppression with replacement is specified
       — the decimal point position.

    b) the second way is to represent all of the numeric character positions by the symbol 'Z'. The result depends upon the value of the data to be stored. If the value is not zero, the result is the same as if the zero suppression with replacement were defined only to the left of the decimal point position. If the value is zero, all character positions of the item will contain the character space.

13.18.39.5 Precedence rules

Where an 'x' appears in Table 11 and in Table 12, additional Syntax Rules or General Rules may apply further restrictions.

FORMAT 1

Table 11, Format 1 picture symbol order of precedence, shows the order of precedence of symbols in a Format 1 picture character-string. An 'x' at an intersection indicates that the symbol(s) at the top of the column may precede (but not necessarily immediately) in character-string-1 the symbol(s) at the left of the row. The currency symbol is indicated by the symbol 'cs'.

The currency symbol when used as a fixed insertion symbol appears in two columns and two rows. The leftmost column and the uppermost row for this symbol represent its use as the first or second symbol in character-string-1. The rightmost column and the lowermost row for this symbol represent its use as the last or penultimate symbol in character-string-1.

The symbol '+' that appears in a column and in a row by itself, represents its use in the exponent part of character-string-1 for a floating-point numeric-edited item.

The symbols '+' and '-' when used as a non-floating insertion symbol appear in two columns and two rows. The leftmost column and the uppermost row for these symbols represent their use as the first symbol in character-string-1. The rightmost column and the lowermost row for these symbols represent their use as the last symbol in character-string-1.

The symbol 'P', the currency symbol when used as a floating insertion symbol, the pair of zero-suppression symbols 'Z' and '*', and the pair of floating insertion symbols '+' and '-' appear in two columns and in two rows in this table. The leftmost column and the uppermost row for these symbols represent their use to the left of the decimal point position. The rightmost column and the lowermost row for these symbols represent their use to the right of the decimal point position.

For the purposes of this table, character-string-1 for a numeric-edited item for a floating-point edited result is considered as two separate strings, the first of which begins with the first symbol and ends with the symbol 'E', and the second of which begins with the symbol 'E' and ends with the last symbol. The presence of symbols preceding the symbol 'E' has no effect on the validity of symbols following the symbol 'E'.

When the DECIMAL-POINT IS COMMA clause is specified, the precedence rules for the symbols comma and period are interchanged.

Table 11 — Format 1 picture symbol order of precedence

Second Symbol		First Symbol																						
		Simple, special, and fixed insertion symbols									Zero-suppression and floating insertion symbols						Other symbols							
		B 0 /	**,**	**.**	**+**	**+ –**	**+ –**	**CR DB**	**cs**	**cs**	**Z ***	**Z ***	**+ –**	**+ –**	**cs**	**cs**	**9**	**A X**	**S**	**V**	**P**	**P**	**1**	**N**	**E**
Simple, special, and fixed insertion symbols	B 0 /	x	x	x		x			x		x	x	x	x	x	x	x	x		x		x		x	
	,	x	x	x		x			x		x	x	x	x	x	x	x			x		x			
	.	x	x			x			x		x		x		x		x								
	+																								x
	+ –																								
	+ –	x	x	x					x	x	x	x			x	x	x			x	x	x			
	CR DB	x	x	x					x	x	x	x			x	x	x			x	x	x			
	cs					x																			
	cs	x	x	x		x					x	x					x			x	x	x			
Zero-suppression and floating insertion symbols	Z *	x	x			x			x		x														
	Z *	x	x	x		x			x		x	x								x		x			
	+ –	x	x						x				x												
	+ –	x	x	x					x				x	x						x					
	cs	x	x			x									x										
	cs	x	x	x		x									x	x				x					
Other symbols	9	x	x	x	x	x			x		x		x		x		x	x	x	x		x			x
	A X	x															x	x							
	S																								
	V	x	x			x			x		x		x		x		x		x		x				
	P	x	x			x			x		x		x		x		x		x		x				
	P					x			x										x	x		x			
	1																						x		
	N	x																						x	
	E	x	x	x		x											x								

FORMAT 2

Table 12, Format 2 picture symbol order of precedence, shows the order of precedence of symbols in a Format 2 picture character-string. An 'x' at an intersection indicates that the symbol at the top of the column may precede (but not necessarily immediately) in character-string-1 the symbol at the left of the row. The currency symbol is indicated by the symbol 'cs'.

Table 12 — Format 2 picture symbol order of precedence

Second Symbol	First Symbol				
	9	**cs**	**.**	**+**	**Z**
9	X	X	X	X	X
cs				X	
.	X	X		X	X
+					
Z		X	X	X	X

13.18.40 PRESENT WHEN clause

1) The PRESENT WHEN clause specifies a condition under which a report section entry will be processed.

2) The PRESENT WHEN clause also enables conditional selection of data description entries by the VALIDATE statement.

13.18.40.1 General format

Format 1 (report-writer):

PRESENT WHEN condition-1

Format 2 (validation):

PRESENT WHEN condition-2

13.18.40.2 Syntax rules

FORMAT 2

1) The PRESENT WHEN clause shall not be specified for a strongly-typed group item or any item subordinate to a strongly-typed group item.

13.18.40.3 General rules

FORMATS 1 AND 2

1) If the PRESENT WHEN clause is specified in a report group description entry, the general rules for format 1 apply, otherwise, the general rules for format 2 apply.

FORMAT 1

2) If a report group contains any entries that have a PRESENT WHEN clause, condition-1 of each PRESENT WHEN clause is evaluated before the processing of any LINE clauses for the report group. The effect of the PRESENT WHEN clause depends on the value of condition-1 as follows:

    a) If condition-1 is true, the corresponding data item is declared to be present and the PRESENT WHEN clause does not affect the processing for this instance of the report group.

    b) If condition-1 is false, the corresponding data item is declared to be absent and the effect on processing is as though the entry were omitted from the description of the report group. If the data description entry is not an elementary entry, all its subordinate data items are also declared to be absent, irrespective of any PRESENT WHEN clauses they may also contain. Furthermore, if the entry is a level-01 entry, the effect on processing is as though the entire report group description were omitted.

3) Within a report group description, any PRESENT WHEN clauses are taken into account when assessing the validity of the arrangement of LINE and COLUMN clauses, the manner in which the report group will be printed and the effect of sum counters, as follows:

    a) The rules for positioning the first line of the report group ignore any LINE clauses specified at the start of the report group where the LINE clauses are associated with absent data items. (See 13.18.34, LINE clause.)

    b) The rules forbidding overlap of absolute lines in the report group are not applied to lines associated with absent data items. (See 13.18.34, LINE clause.)

c) The rules preventing trailing relative lines in the report group from exceeding the report group's lower limit are not applied to lines associated with absent data items. (See 13.18.34, LINE clause.)

d) The page fit test for body groups disregards all lines associated with absent data items. (See 13.18.34, LINE clause.)

e) The rules forbidding overlap of absolute printable items in a report line are not applied to items associated with absent data items. (See 13.18.14, COLUMN clause, general rule 4.)

f) The rules preventing trailing relative printable items in the line from exceeding the page width are not applied to items associated with absent data items. (See 13.18.14, COLUMN clause.)

g) If an entry with a SUM clause is associated with an absent data item, the sum counter is not printed and is not reset to zero.

FORMAT 2

4) The PRESENT WHEN clause takes effect during the execution of a VALIDATE statement that directly or indirectly references the subject of the entry.

5) Condition-2 is evaluated at the beginning of the execution of the format validation stage, with the following two possible results:

   a) If condition-2 is true, the data item that is the subject of the entry is processed during further execution of the VALIDATE statement.

   b) If condition-2 is false, the data item that is the subject of the entry and all data items subordinate to it are not processed at this and all subsequent stages of the execution of the VALIDATE statement.

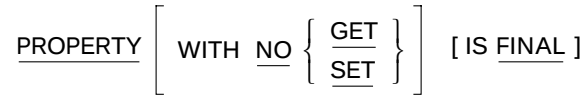
      NOTE If condition-2 is false, the contents of the data item will not be checked unless the data item is redefined.

6) Condition-2 shall not reference any data item that is, or shares any storage with, an operand of a DESTINATION clause appearing later in the description of a data item referred to by the same VALIDATE statement.

13.18.41 PROPERTY clause

The PROPERTY clause indicates that this data item is a property of the object and that GET and/or SET methods are to be generated accordingly.

13.18.41.1 General format



13.18.41.2 Syntax rules

1) The PROPERTY clause may be specified only in the working-storage section of a factory definition or an instance definition.

2) The PROPERTY clause shall not be specified for data items subject to an OCCURS clause.

3) The PROPERTY clause may be specified only for an elementary item whose name does not require qualification for uniqueness of reference.

4) The data-name for the subject of the entry shall not be the same as a property-name defined in a superclass.

   NOTE A property-name may be defined in a superclass either by defining a method or a pair of methods with the PROPERTY phrase or by describing a data description entry with the PROPERTY clause.

5) If the PROPERTY clause is specified in a data item described with the CONSTANT RECORD clause, or in any data item subordinate to a data item described with the CONSTANT RECORD clause, the SET phrase shall be specified.

6) The PROPERTY clause shall not be specified for data items of usage object reference described with an ACTIVE-CLASS phrase.

13.18.41.3 General rules

1) If the GET phrase is not specified, the PROPERTY clause causes a method to be defined for the containing object.

   If the subject of this entry is of class index, object, or pointer, the implicit definition of this method is as follows:

```
METHOD-ID. GET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name data-description.
PROCEDURE DIVISION RETURNING LS-data-name.
par-name.
      SET LS-data-name TO data-name
      EXIT METHOD.
END METHOD.
```

   If the subject of this entry is of category alphanumeric-edited, national-edited, or numeric-edited, the implicit definition of this method is as follows:

```
METHOD-ID.  GET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
```

```
01  LS-data-name data-description.
PROCEDURE DIVISION RETURNING LS-data-name.
par-name.
      MOVE data-name TO LS-data-name (1:)
      EXIT METHOD.
END METHOD.
```

NOTE If the subject of the entry is edited, reference modification of the receiving item as the whole of itself prevents the editing rules from being reapplied to the data.

Otherwise, the implicit definition of this method is as follows:

```
METHOD-ID. GET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name data-description.
PROCEDURE DIVISION RETURNING LS-data-name.
par-name.
      MOVE data-name TO LS-data-name
      EXIT METHOD.
END METHOD.
```

Where LS-data-name has the data description of the subject of the entry with the exception of:

— PROPERTY clauses
— VALUE clauses
— a REDEFINES clause in the description of the subject of the entry.

2) If the SET phrase is not specified, the PROPERTY clause causes a method to be defined for the containing object.

If the subject of this entry is of class index, object, or pointer, the implicit definition of this method is as follows:

```
METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name data-description.
PROCEDURE DIVISION USING LS-data-name.
par-name.
      SET data-name TO LS-data-name
      EXIT METHOD.
END METHOD.
```

If the subject of this entry is of category alphanumeric-edited, national-edited, or numeric-edited, the implicit definition of this method is as follows:

```
METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01  LS-data-name data-description.
PROCEDURE DIVISION USING LS-data-name.
par-name.
      MOVE LS-data-name TO data-name(1:)
      EXIT METHOD.
```

```
END METHOD.
```

NOTE If the subject of the entry is edited, reference modification of the receiving item as the whole of itself prevents the editing rules from being reapplied to the data.

Otherwise, the implicit definition of this method is as follows:

```
METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name data-description.
PROCEDURE DIVISION USING LS-data-name.
par-name.
      MOVE LS-data-name TO data-name
      EXIT METHOD.
END METHOD.
```

Where LS-data-name has the data description of the subject of the entry with the exception of:

- PROPERTY clauses
- VALUE clauses
- a REDEFINES clause in the description of the subject of the entry.

3) If the FINAL phrase is specified, the implicit method definitions generated by the PROPERTY clause shall include the FINAL phrase in their METHOD-ID paragraphs.

13.18.42 RECORD clause

The RECORD clause specifies the number of bytes in a fixed length record, or specifies the range of bytes in a variable-length record. If the number of bytes does vary, the clause specifies the minimum and maximum number of bytes.

13.18.42.1 General format

Format 1 (fixed-length):

RECORD CONTAINS integer-1 CHARACTERS

Format 2 (variable-length):

RECORD IS VARYING IN SIZE [[FROM integer-2] [TO integer-3] CHARACTERS]
[DEPENDING ON data-name-1]

Format 3 (fixed-or-variable-length):

RECORD CONTAINS integer-4 TO integer-5 CHARACTERS

13.18.42.2 Syntax rules

ALL FORMATS

1) If no record description entries are specified in a file description entry for a file that is not a report file, the RECORD clause shall be specified.

FORMAT 1

2) No record description entry for the file may specify a number of bytes greater than integer-1.

FORMAT 2

3) Record descriptions for the file shall describe neither records that contain a lesser number of bytes than that specified by integer-2 nor records that contain a greater number of bytes than that specified by integer-3.

4) Integer-3 shall be greater than integer-2.

5) Data-name-1 shall describe an elementary unsigned integer in the working-storage, local-storage, or linkage section.

6) Integer-2 shall be greater than or equal to zero.

FORMAT 3

7) Integer-4 shall be greater than or equal to zero.

8) Integer-5 shall be greater than integer-4.

13.18.42.3 General rules

ALL FORMATS

1) Each integer in a RECORD clause specifies a record size in terms of bytes.

2) The implicit or explicit RECORD clause specifies the size of records in the record area. The size of records on physical storage media may be different due to control information required by the operating environment. Factors in the source element other than the RECORD clause that may affect the size of records on physical storage medium are the CODE-SET clause and the FORMAT clause. The implementor shall specify the calculations necessary for the user to derive the size for records on the storage medium.

3) The size of each record is specified in terms of the number of bytes required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of bytes in all fixed length elementary items plus the sum of the maximum number of bytes in any occurs-depending table subordinate to the record. Implicit filler positions, if any, are included in the size.

4) If the last data item in the logical record does not end at a byte boundary, the record size includes the entire byte in which that data item ends, and the value of implicit filler bit positions necessary to complete the byte is undefined.

5) If the RECORD clause is not specified, an implicit format 1 or format 2 RECORD clause is assumed to be specified. This implicit RECORD clause is defined by the implementor with the following characteristics:

   a) If format 1 is implied, integer-1 shall be the record size of the largest record description entry in this file description entry.

   b) If format 2 is implied, integer-2 shall be the record size of the smallest record description entry in this file description entry, and integer-3 shall be the largest record description entry in this file description entry. The DEPENDING ON phrase is not specified.

FORMAT 1

6) Format 1 is used to specify fixed-length records. Integer-1 specifies the number of bytes contained in each record in the file.

FORMAT 2

7) Format 2 is used to specify variable-length records. Integer-2 specifies the minimum number of bytes to be contained in any record of the file. Integer-3 specifies the maximum number of bytes in any record of the file.

8) The number of bytes associated with a record description is determined by the sum of the number of bytes in all elementary data items excluding redefinitions and renamings, plus any implicit FILLER due to synchronization. If a table is specified:

   a) The minimum number of table elements described in the record is used in the summation above to determine the minimum number of bytes associated with the record description.

   b) The maximum number of table elements described in the record is used in the summation above to determine the maximum number of bytes associated with the record description.

9) If integer-2 is not specified, the minimum number of bytes to be contained in any record of the file is equal to the least number of bytes described for a record in that file.

10) If integer-3 is not specified, the maximum number of bytes to be contained in any record of the file is equal to the greatest number of bytes described for a record in that file.

11) If data-name-1 is specified, the number of bytes in the record shall be placed into the data item referenced by data-name-1 before any RELEASE, REWRITE, or WRITE statement is executed for the file.

12) If data-name-1 is specified, the execution of a DELETE, RELEASE, REWRITE, START, or WRITE statement or the unsuccessful execution of a READ or RETURN statement does not alter the content of the data item referenced by data-name-1.

13) During the execution of a RELEASE, REWRITE, or WRITE statement, the number of bytes in the record is determined by the following conditions:

    a) If data-name-1 is specified, by the content of the data item referenced by data-name-1.

    b) If data-name-1 is not specified and the record does not contain a variable-occurrence data item, by the number of bytes in the record.

    c) If data-name-1 is not specified and the record does contain a variable-occurrence data item, by the sum of the fixed portion and that portion of the table described by the number of occurrences at the time of execution of the output statement.

14) If the number of bytes in the record to be written is less than integer-2 or greater than integer-3, the following occurs:

    a) If a REWRITE or WRITE statement is being executed, the EC-I-O-LOGIC-ERROR exception condition is set to exist, and the execution of the REWRITE or WRITE statement is unsuccessful.

    b) If a RELEASE statement is being executed, the EC-SORT-MERGE-RELEASE exception condition is set to exist and the execution of the RELEASE statement is unsuccessful.

15) If data-name-1 is specified, after the successful execution of a READ or RETURN statement for the file, the contents of the data item referenced by data-name-1 will indicate the number of bytes in the record just read.

16) If the INTO phrase is specified in the READ or RETURN statement, the number of bytes in the current record that participate as the sending operands in the implicit MOVE statement is determined by the following conditions:

    a) If data-name-1 is specified, by the content of the data item referenced by data-name-1.

    b) If data-name-1 is not specified, by the value that would have been moved into the data item referenced by data-name-1 had data-name-1 been specified.

    If the number of bytes determined as above is zero, the record is a zero-length item.

FORMAT 3

17) It is implementor defined whether format 3 of the RECORD clause produces fixed-length records or variable-length records.

18) When format 3 of the RECORD clause is used, integer-4 and integer-5 refer to the minimum number of bytes in the smallest size record and the maximum number of bytes in the largest size record, respectively. However, in this case, the size of each record is completely defined in the record description entry.

19) If the number of bytes in the logical record to be written is less than integer-4 or greater than integer-5, the following occurs:

    a) If a REWRITE or WRITE statement is being executed, the EC-I-O-LOGIC-ERROR exception condition is set to exist and the execution of the REWRITE or WRITE statement is unsuccessful.

    b) If a RELEASE statement is being executed, the EC-SORT-MERGE-RELEASE exception condition is set to exist and the execution of the RELEASE statement is unsuccessful.

13.18.43 REDEFINES clause

The REDEFINES clause allows the same computer storage area to be described by different data description entries.

13.18.43.1 General format

level-number [entry-name-clause] REDEFINES data-name-2

where entry-name-clause is described in 13.18.19, Entry-name clause

NOTE Level-number and entry-name-clause are shown in the above format to provide context. Level-number and entry-name-clause are not part of the REDEFINES clause.

13.18.43.2 Syntax rules

1) The REDEFINES clause shall immediately follow the entry-name clause; if the entry-name clause is not specified, the REDEFINES clause shall immediately follow the level-number.

2) The level-numbers of data-name-2 and the subject of the entry shall be identical, but shall not be 66 or 88.

3) This clause shall not be specified in level 1 entries in the file section or in any entry subordinate to a file description entry that contains a FORMAT clause.

4) No entry having a level-number numerically lower than the level-number of data-name-2 may occur between the data description entries of data-name-2 and the subject of the entry.

5) The data description entry for data-name-2 shall not contain an OCCURS clause. However, data-name-2 may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause shall not be subscripted. Neither the original definition nor the redefinition shall include an occurs-depending table.

6) Data-name-2 shall not be qualified.

   NOTE If data-name-2 is not unique, no ambiguity of reference exists because of the required placement of the REDEFINES clause.

7) Multiple redefinitions of the same storage area shall each specify as data-name-2 the data-name of the entry that originally defined the area.

8) The storage area required for the subject of the entry shall not be larger than the storage area required for the data item referenced by data-name-2, unless the data item referenced by data-name-2 has been specified with level number 1 and without the EXTERNAL clause.

9) Neither this entry nor any entry subordinate to it shall contain a VALUE clause, unless that VALUE clause is specified in an entry with the level-number 88.

10) The entries giving the new descriptions of the storage area shall follow the entries defining the area of data-name-2, without intervening entries that define new storage areas.

11) Data-name-2 may be subordinate to an entry that contains a REDEFINES clause.

12) The REDEFINES clause shall not be specified for a data item of class object or pointer or a strongly-typed group item.

13) The data description entry for data-name-2 shall not contain the CONSTANT RECORD clause.

14) Data-name-2 shall not be of class object or pointer, a strongly-typed group item, or an item subordinate to a strongly-typed group item.

15) The description of the subject of the entry shall be such that its required alignment is the same as the alignment of the data item referenced by data-name-2.

NOTE This committee draft International Standard places requirements on the alignment and an implementation may place additional requirements on the alignment.

16) Data-name-2 shall not be described with the ANY LENGTH clause.

17) Neither data-name-2 nor the subject of the entry shall be a variable-length group or an any-length elementary item.

NOTE REDEFINES may however be specified in an entry subordinate to a variable-length group or a dynamic-capacity table.

13.18.43.3 General rules

1) Storage association for the subject of the entry starts at the first bit of the data item referenced by data-name-2 and continues over an area sufficient to contain the number of bits required by the data item referenced by the subject of the entry. If the subject of the entry requires more bits than the data item referenced by data-name-2, the storage area allocated for the data item referenced by data-name-2 and the subject of the entry is the number of bits required by the data item referenced by the subject of the entry. The size used for references to the data item referenced by data-name-2 is not changed.

2) When the same storage area is defined by more than one data description entry, the data-name associated with any of those data description entries may be used to reference that storage area.

3) If a REDEFINES clause is specified in the data description entry for a data item subordinate to the operand of a VALIDATE statement, each definition is used by the VALIDATE statement independently of the others, subjecting the same data locations to more than one set of checks. The PRESENT WHEN clause may be used to select one or more of a set of redefinitions, according to the specified conditions.

13.18.44 RENAMES clause

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary items.

13.18.44.1 General format

66 data-name-1 RENAMES data-name-2 [{ THROUGH / THRU } data-name-3] .

NOTE Level-number 66 and data-name-1 are shown in the above format to provide context. Level-number and data-name-1 are not part of the RENAMES clause.

13.18.44.2 Syntax rules

1) Any number of RENAMES entries may be written for a record.

2) All RENAMES entries referring to data items within a given record shall immediately follow the last data description entry of the associated record description entry.

3) Data-name-1 shall not be used as a qualifier. Data-name-1 may be qualified only by the names of the associated level 01, FD, or SD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry.

4) Data-name-2 and data-name-3 shall be names of elementary items or groups of elementary items in the same record, and shall not be the same data-name.

5) Neither data-name-2 nor data-name-3 shall refer to an entry that is described with level-number 1, 66, 77, or 88.

6) Neither data-name-2 nor data-name-3 shall refer to an entry within a record whose data description entry includes the CONSTANT RECORD clause.

7) Data-name-2 and data-name-3 may be qualified.

8) None of the items within the range, including data-name-2 and data-name-3, if specified, shall be of class object or pointer, a strongly-typed group item, an item subordinate to a strongly-typed group item, a variable-length data item, or an occurs-depending table.

9) The words THROUGH and THRU are equivalent.

10) The area described by data-name-2 THROUGH data-name-3 shall define an integral number of bytes.

11) The beginning of the storage area described by data-name-3 shall not precede the beginning of the storage area described by data-name-2. The end of the storage area described by data-name-3 shall follow the end of the storage area described by data-name-2.

NOTE Data-name-3, therefore, shall not be subordinate to data-name-2.

13.18.44.3 General rules

1) When the THROUGH phrase is not specified, all of the data attributes of data-name-2 become the data attributes of data-name-1 and the storage area occupied by data-name-2 becomes the storage area occupied by data-name-1.

2) When the THROUGH phrase is specified, data-name-1 defines an alphanumeric group item that includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item), and concluding with data-name-3 (if data-name-3 is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).

13.18.45 REPORT clause

The REPORT clause identifies the reports that may be written to a report file.

13.18.45.1 General format

```
{ REPORT IS  }
{ REPORTS ARE } { report-name-1 } ...
```

13.18.45.2 Syntax rules

1) Each report-name-1 shall be the subject of a report description entry in the report section of the same source element. The order of appearance of each report-name-1 is not significant.

2) Each report-name-1 may appear in only one REPORT clause.

3) The subject of a file description entry that specifies a REPORT clause may be referenced in the procedure division only by the USE statement, the CLOSE statement, or the OPEN statement with the OUTPUT or EXTEND phrase.

13.18.45.3 General rules

1) The presence of more than one report-name-1 indicates that more than one report may be written to the file.

2) After execution of an INITIATE statement and before the execution of a TERMINATE statement for the same report, no OPEN or CLOSE statements shall be executed that reference the report file.

13.18.46 REQUIRED clause

The REQUIRED clause specifies that in the context of an ACCEPT screen statement, the user shall enter at least one character in the input field.

13.18.46.1 General format

REQUIRED

13.18.46.2 General rules

1) The REQUIRED clause has an effect only during the execution of an ACCEPT statement referencing the screen item.

2) The REQUIRED clause has no effect until the cursor enters a screen item subject to the REQUIRED clause.

3) The effect of the REQUIRED clause is to reject the terminator keystroke and any other cursor-moving keystrokes that would cause the cursor to move to another screen item unless the required termination condition is satisfied. The required termination condition is satisfied:

   — if a screen item is alphanumeric or alphanumeric-edited and contains at least one nonspace character;
   — if a screen item is national or national-edited and contains at least one nonspace character;
   — if a screen item is numeric or numeric-edited and contains a nonzero value;
   — if a screen item is boolean and contains a nonzero value.

4) For fields that are both input and output, the REQUIRED clause may be satisfied by the contents of the identifier or literal referenced in the FROM or USING clause, as well as data keyed by the terminal operator.

5) The REQUIRED clause is not effective if a function key is used to terminate the execution of the ACCEPT statement.

6) The specification of the FULL and REQUIRED clauses together requires that the field be filled entirely before the normal termination key has any effect.

7) If a REQUIRED clause is specified in a group screen item, it applies to each elementary input screen item in that group.

13.18.47 REVERSE-VIDEO clause

The REVERSE-VIDEO clause specifies that the screen item is to be displayed by exchanging the foreground and background colors that would otherwise be in effect.

13.18.47.1 General format

REVERSE-VIDEO

13.18.47.2 General rules

1) If the REVERSE-VIDEO clause is specified at group level, it applies to each elementary screen item in that group.

2) When the REVERSE-VIDEO clause is specified, the screen item will be displayed so that the characters that constitute the screen item will be shown with the foreground and background colors being exchanged when the screen item is referenced in an ACCEPT screen or a DISPLAY screen statement. For monochrome displays, the reverse-video attribute is used.

13.18.48 SAME AS clause

The SAME AS clause specifies that a data-name has the same description as that specified by another data description entry.

13.18.48.1 General format

SAME AS data-name-1

13.18.48.2 Syntax rules

1) Data-name-1 may be qualified.

2) A data description entry that specifies the SAME AS clause shall not be immediately followed by a subordinate data description entry or level 88 entry.

3) Neither the description of data-name-1 nor the description of any data items subordinate to the subject of the entry shall directly or indirectly contain a SAME AS clause that references the subject of the entry or any group item to which this entry is subordinate.

4) The description of data-name-1, including its subordinate data items, shall not contain a TYPE clause that references the record to which this entry is subordinate.

5) The description of data-name-1 shall not contain an OCCURS clause. However, items subordinate to data-name-1 may contain OCCURS clauses.

6) When the SAME AS clause is specified in the file section, the description of data-name-1, including its subordinate data items, shall not contain a data item described with a USAGE OBJECT REFERENCE clause.

7) Data-name-1 shall reference an elementary item or a level 1 group item described in the file, working-storage, local-storage, or linkage section.

8) If the subject of the entry is a level 77 item, data-name-1 shall reference an elementary item.

9) A group item to which the subject of the entry is subordinate shall not contain a GROUP-USAGE, SIGN, or USAGE clause.

10) The description of data-name-1 shall not contain a CONSTANT RECORD clause.

13.18.48.3 General rules

1) The effect of the SAME AS clause is as though the data description identified by data-name-1 had been coded in place of the SAME AS clause, excluding the level-number, name, and the CONSTANT RECORD, EXTERNAL, GLOBAL, REDEFINES, and SELECT WHEN clauses specified for data-name-1; level numbers of subordinate items may be adjusted as described in general rule 2.

2) If data-name-1 describes a group item:

   a) the subject of the entry is a group whose subordinate elements have the same names, descriptions, and hierarchy as the subordinate elements of data-name-1,

   b) the level-numbers of items subordinate to that group are adjusted, if necessary, to preserve the hierarchy of data-name-1,

   c) level-numbers in the resulting hierarchy may exceed 49.

   NOTE If alignment according to 8.5.1.5.4, Item alignment for increased object-code efficiency, inserts implicit fillers in either data-name-1 or the subject of the entry, the alignment of the corresponding data items may differ.

3) If an alphanumeric group item or strongly-typed group item to which data-name-1 is subordinate contains a USAGE clause, the effect is as though that USAGE clause had been specified for the subject of the entry.

4) If a group item to which data-name-1 is subordinate contains a GROUP-USAGE clause and the subject of the entry is a group item, the effect is as if that GROUP-USAGE clause had been specified for the subject of the entry.

5) If an alphanumeric group item, national group item, or strongly-typed group item to which data-name-1 is subordinate contains a SIGN clause, the effect is as though that SIGN clause had been specified for the subject of the entry.

13.18.49 SECURE clause

The SECURE clause prevents data entered from the keyboard or contained in the screen item from appearing on the screen at the screen location that corresponds to the screen item for which it is specified.

13.18.49.1 General format

SECURE

13.18.49.2 General rules

1) If a SECURE clause is specified at the group level, it applies to each elementary input screen item in that group.

2) The SECURE clause has an effect only during the execution of an ACCEPT statement referencing the screen item.

3) The effect of the SECURE clause is to prevent data corresponding to a screen item that has been specified with the SECURE clause from being displayed on the screen. During the execution of an ACCEPT statement, the cursor will appear at the screen location that corresponds to an input screen item, but any data keyed by the terminal operator will not be displayed. For fields that are both input and output, the contents of the screen location of the screen item prior to the execution of the ACCEPT screen statement remain unchanged and are unchangeable by the terminal operator.

4) It is implementor-defined whether the cursor moves as data is entered into a field for which the SECURE clause is specified.

13.18.50 SELECT WHEN clause

The SELECT WHEN clause specifies a condition-name condition under which a record description entry is associated with a record during input and output operations.

13.18.50.1 General format

```
SELECT WHEN { condition-name-1 }
            { OTHER           }
```

13.18.50.2 Syntax rules

1) A SELECT WHEN clause may be specified only at the 01 level of a record description entry in the file, linkage, local-storage, or working-storage section.

2) If a SELECT WHEN clause is specified for any record description entry associated with a given file-name, then a SELECT WHEN clause shall be specified for all record description entries associated with that file-name.

3) A given condition-name-1 shall be specified in the SELECT WHEN clause of only one record description entry associated with a given file-name.

4) For a given file description entry, all instances of condition-name-1 shall be associated with a conditional variable defined at the same position in the record, with the same usage, and with the same size.

5) If a FORMAT clause is specified for the file description entry, the conditional variable associated with condition-name-1 shall be the first elementary data item in the record description entry.

6) The OTHER phrase may be specified only in the last record description entry associated with a given file.

13.18.50.3 General rules

1) For a given file-description entry, any SELECT WHEN clauses of associated record description entries are evaluated for READ statements, for REWRITE statements with the FILE phrase, and for WRITE statements with the FILE phrase, if the associated file description entry also contains a CODE-SET clause or FORMAT clause.

2) SELECT WHEN clauses of record description entries specified in the working-storage section, local-storage section, or the linkage section have no effect. SELECT WHEN clauses of record description entries specified in the file section have no effect for statements other than READ statements, REWRITE statements with the FILE phrase, and WRITE statements with the FILE phrase, when those statements reference the associated file.

3) The condition-name condition specified in a SELECT WHEN clause is evaluated for each record description entry, in the order the record description entries are written, until the evaluation is true. When the evaluation is true, the associated record description entry is selected for use with the CODE-SET and FORMAT clauses. The evaluation is always true when the OTHER phrase is specified.

4) If no record description entry is selected, the input-output operation is unsuccessful and the I-O status is set to a value indicating a record identification failure. (See 9.1.12, I-O status.)

5) The implementor shall specify whether a SELECT WHEN clause takes effect for READ statements, for REWRITE statements with the FILE phrase, or for WRITE statements with the FILE phrase, when the associated file-description entry contains neither a CODE-SET clause nor a FORMAT clause.

NOTE This permits code-set conversion based on record layout in circumstances that the implementor defines; a CODE-SET clause need not be present in the source element in this case.

13.18.51 SIGN clause

The SIGN clause specifies the position and the mode of representation of the operational sign.

13.18.51.1 General format

[SIGN IS] { LEADING / TRAILING } [SEPARATE CHARACTER]

13.18.51.2 Syntax rules

1) The SIGN clause may be specified only for:

   — a numeric data or screen description entry whose picture character-string contains the symbol 'S'
   — a numeric report group description entry whose picture character-string contains the symbol 'S'
   — an alphanumeric group item, national group item, or strongly-typed group item.

2) The usage of an elementary item for which the SIGN clause is specified shall be display or national.

3) If the CODE-SET clause is specified in a file description entry, any signed numeric data description entries associated with that file description entry shall be described with the SIGN IS SEPARATE clause.

13.18.51.3 General rules

1) The SIGN clause specifies the position and the mode of representation of the operational sign for the numeric item to which it applies, or for each numeric item subordinate to the group to which it applies. The SIGN clause applies only to numeric items whose picture character-string contains the symbol 'S'.

2) If a SIGN clause is specified in a group item subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate group item takes precedence for that subordinate group item.

3) If a SIGN clause is specified in an elementary numeric item subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in that elementary entry takes precedence for that elementary entry.

4) A numeric item whose picture character-string contains the symbol 'S', and to which no SIGN clause applies, has an operational sign. Neither the representation nor the position of that operational sign is specified by the symbol 'S'. The implementor shall specify the position and representation of the operational sign. General rules 5 and 6 do not apply to such signed numeric items.

5) If the SEPARATE CHARACTER phrase is not specified, then:

   a) The operational sign is presumed to be associated with the leading (or, respectively, trailing) digit position of the data item to which it applies.

   b) The implementor defines what constitutes valid signs for data items.

6) If the SEPARATE CHARACTER phrase is specified, then:

   a) The operational sign is presumed to be the leading (or, respectively, trailing) character position of the data item to which it applies; this character position is not a digit position.

   b) The operational signs for positive and negative are the basic special characters '+' and '–', respectively.

7) Each numeric item whose picture character-string contains the symbol 'S' is a signed item. If a SIGN clause applies to such an item and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

13.18.52 SOURCE clause

The SOURCE clause identifies a data item or expression to be moved automatically to a printable item.

13.18.52.1 General format

```
{ SOURCE IS  }{ identifier-1             } ... [ rounded-phrase ]
{ SOURCES ARE }{ arithmetic-expression-1 }
```

where rounded-phrase is described in 14.7.3, ROUNDED phrase.

13.18.52.2 Syntax rules

1) SOURCE and SOURCES are synonyms.

2) If identifier-1 is specified without the ROUNDED phrase, identifier-1 shall be described such that a MOVE statement is valid with identifier-1 as the sending operand and the printable item as the receiving operand.

3) If arithmetic-expression-1 or the ROUNDED phrase is specified, the entry shall define either a numeric data item or a numeric-edited data item.

4) Identifier-1 specifies a data item defined in any section of the data division. If identifier-1 specifies a report section item, it shall be a report counter identifier or a sum counter defined in the current report. This same syntax rule applies to any identifier appearing in arithmetic-expression-1.

5) If identifier-1 is specified with the ROUNDED phrase, it is considered to be an arithmetic-expression.

6) If the SOURCE clause has more than one operand, the entry shall be a repeating entry or shall be subordinate to a repeating entry, and the number of operands of the SOURCE clause shall be equal to the number of repetitions of the repeating entry or the same number multiplied by the number of repetitions of any number of successive repeating entries at higher levels than the repeating entry.

7) If the SOURCE clause has more than one operand of which at least one is an arithmetic-expression, each operand shall be enclosed in parentheses.

8) Identifier-1 shall not reference a variable-length group.

13.18.52.3 General rules

1) If identifier-1 is specified without the ROUNDED phrase, it specifies the sending operand of an implicit MOVE statement in which the data item referenced by identifier-1 is moved to the printable item.

2) Arithmetic-expression-1 specifies the operand of an implicit COMPUTE statement that is executed implicitly whenever the associated item is printed. If the ROUNDED phase is specified, the implicit COMPUTE statement has the corresponding ROUNDED phrase.

3) If the entry containing the SOURCE clause contains no COLUMN clause and therefore defines an unprintable item, the SOURCE clause causes no action, except where the entry is referred to by means of a SUM clause. In all other cases, a MOVE or COMPUTE statement is implicitly executed before the associated report line is printed. If identifier-1 is specified without the ROUNDED phrase, the statement executed is the MOVE statement defined in general rule 1 above. Otherwise, the statement executed is the COMPUTE statement defined in general rule 2 above.

4) If the SOURCE clause has more than one operand, successive operands are assigned to successive repeating printable items, horizontally and then vertically, as applicable, in that hierarchic order. If no further operands remain, assignment begins again from the first operand. If any of the printable items are suppressed as a result

of a PRESENT WHEN clause or an OCCURS clause with a DEPENDING phrase, SOURCE operands are nevertheless assigned to them, even though they are not printed.

13.18.53 SUM clause

The SUM clause specifies one or more data items that are to be totalled to provide the value of the associated elementary report item.

13.18.53.1 General format

```
{ SUM OF { data-name-1          } ... [ UPON { data-name-2 } ... ] } ...
         { identifier-1         }
         { arithmetic-expression-1 }

[ RESET ON { data-name-3 } ] [ rounded-phrase ]
           { FINAL       }
```

where rounded-phrase is described in 14.7.3, ROUNDED phrase.

13.18.53.2 Syntax rules

1) Each data-name-1, identifier-1 or arithmetic-expression-1 is an addend. The whole clause is referred to as a SUM clause even though the SUM keyword may appear more than once.

2) The category of the subject of the entry shall be valid as the category of a receiving operand in a MOVE statement for a sending operand of the category numeric.

3) The ROUNDED phrase may be specified in the SUM clause only if the COLUMN clause is specified for the subject of the entry.

4) Data-name-1 shall be the name of a numeric data item in the report section. It may be qualified. If it is associated with an OCCURS clause, it shall be specified without the subscripting normally required. When data-name-1 is specified, the following rules also apply:

   a) The UPON phrase shall not be specified.

   b) If data-name-1 is specified in the same report group description as the subject of the entry, data-name-1 shall be a repeating item, as defined in 13.15, Report group description entry, general rule 3, and subject to at least one more level of repetition than the subject of the entry.

   c) If data-name-1 is specified in a different report group description than the subject of the entry, data-name-1 either shall not be a repeating item or shall be a repeating item that is subject to at least the same number of levels of repetition as the subject of the entry.

   d) The maximum number of repetitions of data-name-1 and the subject of the entry shall be equal at each corresponding level taken in order beginning with the lowest level of nesting. Levels superordinate to those corresponding levels may specify any number of repetitions.

   e) Any chain of reference shall terminate at an entry that does not contain a SUM clause referring to a data-name-1 defined in the report section.

   f) If data-name-1 specifies an entry in a report group description other than the current report group description, only the following combinations of report types of each report group are permitted:

      The current report group may be a control footing and data-name-1 may be defined in a detail or in a control footing associated with a lower level of control.

The current report group may be a detail and data-name-1 may be defined in a different detail.

The current report group may be a report footing and data-name-1 may be defined in any other report group other than a report heading.

The current report group may be a page footing and data-name-1 may be defined in any body group.

g) If data-name-1 specifies an entry in a different report description, there are no restrictions on the combinations of report types of each report group.

5) If the addend is identifier-1, it shall specify a numeric data item not defined in the report section.

6) If the addend is arithmetic-expression-1, any identifiers it contains may reference entries in any section of the data division other than the report section.

7) Data-name-2 shall be the name of a detail. It may be qualified by a report-name.

8) Data-name-3 may be qualified and reference modified. Data-name-3 or FINAL shall be an operand of the CONTROL clause of the current report description. If data-name-3 is reference modified, leftmost-position and length shall be integer literals. If the current report group is a control footing, its level of control shall be a lower level than that of data-name-3.

9) Within a report description entry, if the keyword SUM is preceded by the keyword FUNCTION, SUM is a reference to the SUM intrinsic function. Otherwise, SUM refers to the report writer SUM clause, even when SUM is implicitly or explicitly included as an intrinsic-function-name in the intrinsic format of a function-specifier in the REPOSITORY paragraph.

13.18.53.3 General rules

1) Each entry containing a SUM clause establishes an independent sum counter and size error indicator. The sum counter is an internal register that behaves as a data item of the category numeric. The number of decimal digits in the sum counter, both integral and fractional, is derived from the corresponding number of digits, excluding insertion editing characters, in the PICTURE clause of the entry containing the SUM clause. The sum counter is signed, whether or not the corresponding PICTURE clause has an operational sign.

2) The sum counter is set to zero and its associated size error indicator is unset when the INITIATE statement for the current report is executed. Subsequently, the sum counter is reset to zero and the size error indicator is unset at the end of the processing of the report group in which it is printed or, if the RESET phrase is specified, at the end of the processing of the control footing for the specified level of control. If no such control footing is defined, it is assumed to be present and to consist of a 01-level entry alone.

3) The current content of each addend is implicitly added into the sum counter during execution of GENERATE and TERMINATE statements at specific times defined below. The adding is consistent with the general rules of the ADD statement with the ON SIZE ERROR phrase or, in the case of an arithmetic expression, the COMPUTE statement with the ON SIZE ERROR phrase. Any algebraic sign of the addend is taken into account. Each addition is tested for size error; if a size error occurs, the EC-REPORT-SUM-SIZE exception condition is set to exist. (See 14.7.6, Arithmetic statements.)

4) If the entry also contains a COLUMN clause, the sum counter acts as a source data item. If the ROUNDED phrase is specified in the SOURCE clause, the content of the sum counter is computed according to the general rules for the COMPUTE statement with the ROUNDED phrase, as described under 13.18.52, SOURCE clause. If the associated size error indicator is not set, the content of the sum counter is moved, according to the general rules of the MOVE statement, to the printable item for printing. If the associated size error indicator is set, an EC-REPORT-SUM-SIZE exception condition is set to exist and the printable item is filled with spaces.

5) If a data-name immediately follows the level number in the entry containing the SUM clause, the data-name is the name of the sum counter, not the name of the associated printable item, if any.

6) If data-name-1 specifies an item whose entry contains a SUM clause, the value added is that of the corresponding sum counter. The additions necessary to compute its value are completed before the adding of the operand into the current sum counter. If data-name-1 specifies an item whose entry has a SOURCE or VALUE clause, the value added is that of the operand of the SOURCE or VALUE clause.

7) The times at which addition takes place are defined as follows:

   a) If data-name-1 is the name of an entry in a different report group description, adding takes place when the report group description containing data-name-1 is processed.

   b) If data-name-1 is the name of an entry in the current report group description, adding takes place during the processing of the current report group before any of the report group's lines are printed.

   c) If the addend is identifier-1 or arithmetic-expression-1, adding takes place either:

      1. if no UPON phrase is specified, whenever any GENERATE statement is executed for the current report or any detail defined for the current report, or

      2. If an UPON phrase is specified, whenever any GENERATE statement is executed for a detail referenced by the UPON phrase.

         NOTE Ambiguities can arise if FUNCTION ALL INTRINSIC is specified in the REPOSITORY paragraph and the SUM intrinsic function is used as an operand in arithmetic-expression-1. Avoiding ALL in this context in favor of a list of specific functions used in the program (excluding SUM) eliminates the possibility of such ambiguities. Simply identifying the references to the SUM intrinsic function by the use of the FUNCTION keyword while continuing to specify FUNCTION ALL INTRINSIC in the REPOSITORY paragraph may not be sufficient to resolve such ambiguities.

   If two or more instances of data-name-1 or identifier-1 specify the same data item, this data item is added into the sum counter as many times as data-name-1 or identifier-1 is referenced in the SUM clause. It is permissible for the UPON phrase to contain two or more instances of data-name-2 that specify the same detail. When a GENERATE statement for such a detail is executed, the adding takes place as many times as data-name-2 appears in the UPON phrase.

8) If the addend is data-name-1 and data-name-1 is a repeating item, repetitions of the addend are either all added into the same sum counter or are each added into a different corresponding occurrence of the sum counter, according to the following rules:

   a) If the addend and the sum counter are subject to the same number of levels of repetition, each occurrence of the addend is added into the corresponding occurrence of the sum counter.

   b) If the addend is subject to a greater number of levels of repetition than the sum counter, the number of levels of repetition of the addend is effectively reduced to that of the sum counter by forming the total of a complete table of occurrences of the addend at one or more levels into each occurrence of the sum counter. The levels at which these totals are formed are those of the highest level of repetition of the addend, excluding any OCCURS, multiple LINE, or multiple COLUMN clauses to which the addend and the entry containing the SUM clause are both subject.

9) If the SUM clause specifies more than one addend, the result is the same as when all the addends were summed separately according to the above rules and the results added together.

10) If the entry is associated with an absent data item as a result of a PRESENT WHEN clause or an OCCURS clause with the DEPENDING phrase, the corresponding sum counter is not printed and is not reset to zero for the current instance of the report group.

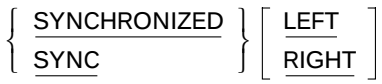
11) If the operand is data-name-1 and is declared to be absent as a result of a PRESENT WHEN clause or an OCCURS clause with the DEPENDING phrase, data-name-1 is not added into the sum counter during the processing that instance of the report group in which data-name-1 is defined.

12) It is permissible for procedure division statements to alter the content of sum counters.

13.18.54 SYNCHRONIZED clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundaries of the computer memory (see 8.5.1.5.4, Item alignment for increased object-code efficiency).

13.18.54.1 General format



13.18.54.2 Syntax rules

1) The SYNCHRONIZED clause may be specified only for an elementary item.

2) SYNC is an abbreviation for SYNCHRONIZED.

13.18.54.3 General rules

1) This clause specifies that the subject data item is to be aligned in the computer such that no other data item occupies any of the byte positions between the leftmost and rightmost natural boundaries delimiting this data item. If the number of bytes required to store this data item is less than the number of bytes between those natural boundaries, the unused bytes or portions thereof shall not be used for any other data item. The unused bytes are not included in the bytes redefined when the elementary item is the object of a REDEFINES clause. Such unused bytes, however, are included in:

    a) The size of any alphanumeric group item, bit group item, national group item, or strongly-typed group item that contains the subject of the entry; and

    b) The number of bytes allocated when any such group item is the object of a REDEFINES clause.

2) SYNCHRONIZED not followed by either RIGHT or LEFT specifies that the elementary item is to be positioned between natural boundaries in such a way as to effect efficient utilization of the elementary data item. The specific positioning is determined by the implementor.

3) SYNCHRONIZED LEFT specifies that the elementary item is to be positioned such that it will begin at the left byte of the natural boundary in which the elementary item is placed.

4) SYNCHRONIZED RIGHT specifies that the elementary item is to be positioned such that it will terminate on the right byte of the natural boundary in which the elementary item is placed.

5) Any adjustment in storage position resulting from the SYNCHRONIZED clause does not affect the size of the synchronized data item.

6) If the data description of an item contains an operational sign and any form of the SYNCHRONIZED clause, the sign of the item appears in the sign position explicitly or implicitly specified by the SIGN clause.

7) When the SYNCHRONIZED clause is specified in the data description entry that contains an OCCURS clause, or in the data description entry of a data item subordinate to the data description entry that contains an OCCURS clause, then:

    a) The SYNCHRONIZED clause applies to each occurrence of the data item.

    b) Any implicit FILLER generated for other data items within that same table is generated for each occurrence of those data items (see general rule 8b).

8) The effect of this clause is defined by the implementor and in addition to the other general rules for the SYNCHRONIZED clause, the implementor shall specify how elementary items associated with this clause are handled regarding:

    a) The format on the external media of records or groups containing elementary items whose data description contains the SYNCHRONIZED clause.

    b) Any necessary generation of implicit FILLER, if the elementary item immediately preceding an item containing the SYNCHRONIZED clause does not terminate at an appropriate natural boundary. Such automatically generated FILLER positions are included in:

        1. The size of any group item to which the FILLER item belongs; and

        2. The number of bytes allocated when the group item of which the FILLER item is a part appears as the object of a REDEFINES clause.

9) An implementor may optionally specify automatic alignment for any internal data representations except for bit data items and, within a record, data items described with usage display or national. A record itself may be automatically synchronized.

10) Any rules for synchronization of the records of a file, as this affects the synchronization of elementary items, will be specified by the implementor.

13.18.55 TO clause

The TO clause identifies the destination of the data in an ACCEPT screen statement.

13.18.55.1 General format

TO identifier-1

13.18.55.2 Syntax rules

1) The category of identifier-1 shall be a permissible category as a receiving operand in a MOVE statement where the sending operand has the same PICTURE clause as the subject of the entry.

2) Identifier-1 shall be defined in the file, working-storage, local-storage, or linkage section. Identifier-1 shall not specify a variable-length group.

3) If the subject of this entry is subject to an OCCURS clause, identifier-1 shall be specified without the subscripting normally required. Additional requirements are specified in 13.18.37, OCCURS clause, syntax rule 13.

13.18.55.3 General rules

1) The subject of the entry is an input screen item.

2) The TO clause has an effect only during execution of an ACCEPT screen statement referencing the screen item.

13.18.56 TYPE clause

The TYPE clause in a data description entry specifies that the data description of the entry is specified by a type declaration.

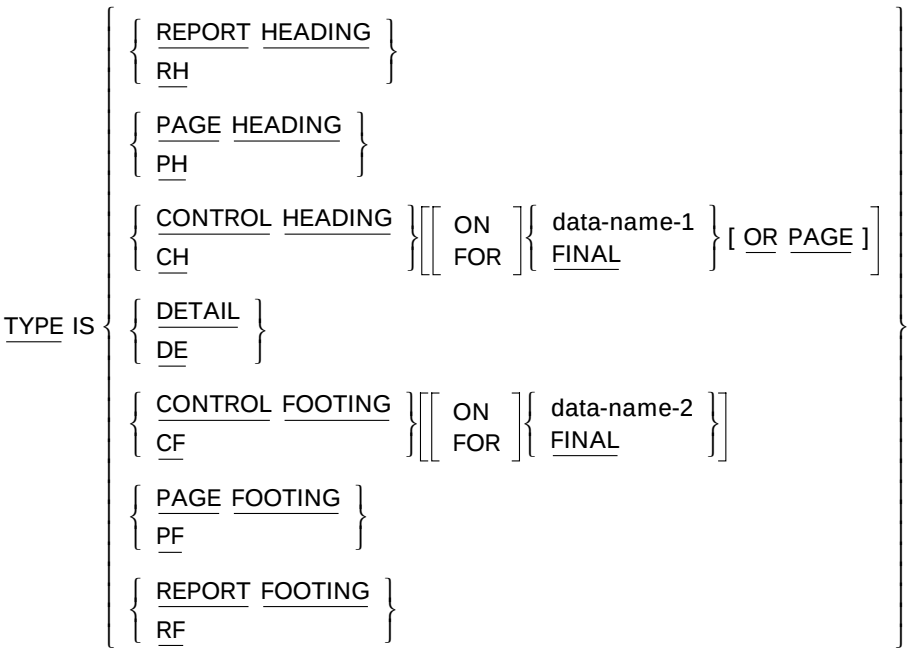
The TYPE clause in a report group description entry identifies the circumstances under which a report group will be printed.

13.18.56.1 General format

Format 1 (type-name):

```
TYPE TO type-name-1
```

Format 2 (report-group):



13.18.56.2 Syntax rules

FORMAT 1

1) The description of any data item subordinate to type-name-1 shall not contain a SAME AS clause that references the subject of this entry or any group item to which this entry is subordinate.

2) A data description entry in which aTYPE clause is specified shall not be followed immediately by a subordinate data description entry or a level 88 entry.

3) If type-name-1 is described with the STRONG phrase, the subject of the entry shall not be renamed in whole or in part.

4) If type-name-1 is described with the STRONG phrase, the subject of the entry shall not be implicitly or explicitly redefined in whole or in part.

5) No group item to which the subject of the entry is subordinate shall contain a GROUP-USAGE, SIGN, or USAGE clause.

6) If type-name-1 is described with the STRONG phrase, the subject of the entry shall be specified only with level number 1 or be subordinate to a type declaration that includes the STRONG phrase.

7) If the subject of the entry is a level 77 item, type-name-1 shall be an elementary item.

8) When theTYPE clause is specified in the file section, the description of type-name-1, including its subordinate data items, shall not contain a data item described with a USAGE OBJECT REFERENCE clause.

FORMAT 2

9) RH is an abbreviation for REPORT HEADING.

   PH is an abbreviation for PAGE HEADING.

   CH is an abbreviation for CONTROL HEADING.

   DE is an abbreviation for DETAIL.

   CF is an abbreviation for CONTROL FOOTING.

   PF is an abbreviation for PAGE FOOTING.

   RF is an abbreviation for REPORT FOOTING.

10) Data-name-1 and data-name-2 may be qualified and reference modified. If data-name-1 or data-name-2 is reference modified, leftmost-position and length shall be integer literals. Each data-name-1, data-name-2, and FINAL, if specified, shall be the same as one of the operands of the CONTROL clause of the corresponding report description entry.

11) Following CONTROL HEADING or CONTROL FOOTING, data-name-1, data-name-2, or FINAL may be omitted only if the CONTROL clause in the corresponding report description entry contains exactly one operand.

12) PAGE HEADING and PAGE FOOTING and the OR PAGE phrase are allowed only if a PAGE clause that defines the page limit is specified in the report description entry.

13) REPORT HEADING, PAGE HEADING, REPORT FOOTING, and PAGE FOOTING may each appear no more than once in any given report description.

14) At most one CONTROL HEADING and at most one CONTROL FOOTING may be defined for each control data item or FINAL of the CONTROL clause for any given report. Either or both of CONTROL HEADING and CONTROL FOOTING may be omitted for any given level of control.

15) Groups of type DETAIL, CONTROL HEADING, and CONTROL FOOTING are referred to as body groups. Each report description shall include at least one body group.

16) If no GENERATE data-name statements are specified in the procedure division, the report description need not contain a DETAIL.

13.18.56.3 General rules

FORMAT 1

1) The TYPE clause specifies that the data description of the subject of the entry is specified by type-name-1. The effect of the TYPE clause is as though the data description identified by type-name-1 had been coded in place of the TYPE clause, excluding the level-number, name, alignment, and the GLOBAL, SELECT WHEN, and TYPEDEF clauses specified for type-name-1; level numbers of subordinate items may be adjusted as described in general rule 2.

2) If type-name-1 describes a group item:

   a) the subject of the entry is a group whose subordinate elements have the same names, descriptions, and hierarchy as the subordinate elements of type-name-1,

   b) the level-numbers of items subordinate to that group are adjusted, if necessary, to preserve the hierarchy of type-name-1,

   c) level-numbers in the resulting hierarchy may exceed 49,

   d) the subject of the entry is aligned as though it were a level 1 item.

3) If a VALUE clause is specified in the data description of the subject of the entry, the content of the literal associated with that VALUE clause is used for the initial value associated with the subject of the entry. When the description of type-name-1 includes an implicit PICTURE clause derived from a VALUE clause, that implicit PICTURE clause becomes part of the description of the subject of the entry.

4) If a BASED clause is specified in the data description of the subject of the entry, that BASED clause applies and any BASED clause specified in the description of type-name-1 is ignored for this entry.

FORMAT 2

5) Report groups are printed only during the execution of a GENERATE or a TERMINATE statement. Detail report groups are printed when referenced explicitly in a GENERATE statement. All other report groups are printed implicitly according the general rules that follow.

6) The conditions under which a given report group is printed depend on its type as follows:

   a) The report heading, if defined, is printed as the first report group in the report, when the chronologically first GENERATE statement for the report, if any, is executed.

   b) The page heading, if defined, is printed immediately before, and on the same page as, the chronologically first body group to be printed for the report, and subsequently as the first report group in each new page whenever a page advance takes place, except when the report group about to be printed is a report footing on a page by itself.

   c) Each control heading without the OR PAGE phrase, wherever defined, is printed automatically, in either of the following events:

      1. when the chronologically first GENERATE statement following an INITIATE statement for the report is executed, in order of control levels from highest to lowest,

      2. immediately preceding any detail printed as the result of the execution of a GENERATE statement when a control break has been detected, in order of control levels from the level of the control break down to the lowest.

      The OR PAGE phrase causes the associated control heading to be printed in addition after each page advance, following any page heading, provided that the page advance did not take place just before the printing of a control footing at a lower control level.

   d) A detail is printed as the result of the execution of an explicit GENERATE statement that references it.

   e) Each control footing, wherever defined, is printed automatically in either of the following events:

      1. when a GENERATE statement is executed where a control break has been detected, preceding any control heading and detail groups, in order of controls from the lowest up to the level of the control break,

2. when the TERMINATE statement is executed for the report, provided that at least one GENERATE statement has been executed after the chronologically last INITIATE statement for the report, in order of controls from lowest to highest.

f) The page footing, if defined, is printed as the last report group on each page of the current report, except in the following cases:

1. on the first page, if it is occupied only by a report heading group;

2. on the last page, if it is occupied only by a report footing group;

If a report footing is defined and is not on a page by itself, the page footing on the last page is immediately followed by the report footing.

g) The report footing, if defined, is printed, as the very last report group in the report, when a TERMINATE for the report is executed, provided that at least one GENERATE statement has been executed for the report since the chronologically last INITIATE statement was executed for the report.

7) The upper limit is defined to be the uppermost permitted line on the page that may be occupied by the report group's first line. It is calculated as follows:

a) The upper limit for a report heading or a page heading where no report heading appears on the same page is the line given by the HEADING integer.

b) The upper limit for a page heading where a report heading appears on the same page is the line following the last line of the report heading.

c) The upper limit for a body group, if there is no control heading defined in the report with the OR PAGE phrase, is the line given by the FIRST DETAIL integer.

d) If there is at least one control heading defined in the report with the OR PAGE phrase, the upper limit for a body group is determined as follows:

1. If the body group is a control heading at the same or a higher control level than the highest-level control heading that has an OR PAGE phrase, the upper limit is the line given by the FIRST DETAIL integer.

2. If the body group is a control heading at a lower control level than the highest-level control heading that has an OR PAGE phrase, the upper limit is the line following the last line of the next higher-level control heading.

3. If the body group is a detail, the upper limit is the line following the last line of the lowest-level control heading that has an OR PAGE phrase.

4. If the body group is a control footing, the upper limit is the line following the last line of the lowest-level control heading with an OR PAGE phrase at the same level as the control footing, or higher. If no such control heading is defined, the upper limit is the FIRST DETAIL integer.

e) The upper limit for a page footing is the line number obtained by adding 1 to the FOOTING integer.

f) The upper limit for a report footing that appears on a page by itself is the line given by the HEADING integer.

g) The upper limit for a report footing that does not appear on a page by itself is the line following the last line of the page footing, if specified, or the line number obtained by adding 1 to the FOOTING integer, if no page footing is defined for the report.

8) The lower limit is defined to be the lowermost permitted line on the page that may be occupied by the report group's last line. It is calculated as follows:

   a) The lower limit for a report heading that appears on a page by itself is the page limit.

   b) The lower limit for a report heading that does not appear on a page by itself is the line preceding the first line of the page heading. If no page heading is defined for the report, the lower limit is the value obtained by subtracting 1 from the FIRST DETAIL integer.

   c) The lower limit for a page heading is the line number obtained by subtracting 1 from the FIRST DETAIL integer.

   d) The lower limit for a control heading is the line given by the LAST CONTROL HEADING integer.

   e) The lower limit for a detail is the line given by the LAST DETAIL integer.

   f) The lower limit for a control footing is the line given by the FOOTING integer.

   g) The lower limit for a page footing or report footing is the page limit.

13.18.57 TYPEDEF clause

The TYPEDEF clause specifies that the data description entry is a type declaration.

NOTE A data description entry declared at level-number 1 that includes a TYPEDEF clause is not a record description entry.

13.18.57.1 General format

IS TYPEDEF [STRONG]

13.18.57.2 Syntax rules

1) If the subject of the entry is an elementary item, the STRONG phrase shall not be specified.

2) The description of the subject of the entry, including its subordinate items, shall not contain a TYPE clause that directly or indirectly references this type definition.

13.18.57.3 General rules

1) If the TYPEDEF clause is specified, the data description entry is a type declaration. The data-name specified in the entry is the type-name. Subordinate data description entries, condition-name entries, and RENAMES clauses are part of the type declaration of that type. The data-names of the items described in these subordinate entries may be referenced only as subordinate items of groups defined using the type-name. If there is more than one such group, qualification with the name of the group is necessary. If there is no such group, any reference to a name that is the same as a data-name in the subordinate entry is a reference to another resource, if there is another resource with that name, or is an invalid reference.

2) A type declaration has no storage associated with it.

3) The GLOBAL clause applies to the scope of the type-name. All other data description clauses and subordinate data descriptions are assumed by data defined using the type-name.

13.18.58 UNDERLINE clause

The UNDERLINE clause specifies that each character of the field is underlined when it is displayed on the screen.

13.18.58.1 General format

UNDERLINE

13.18.58.2 General rules

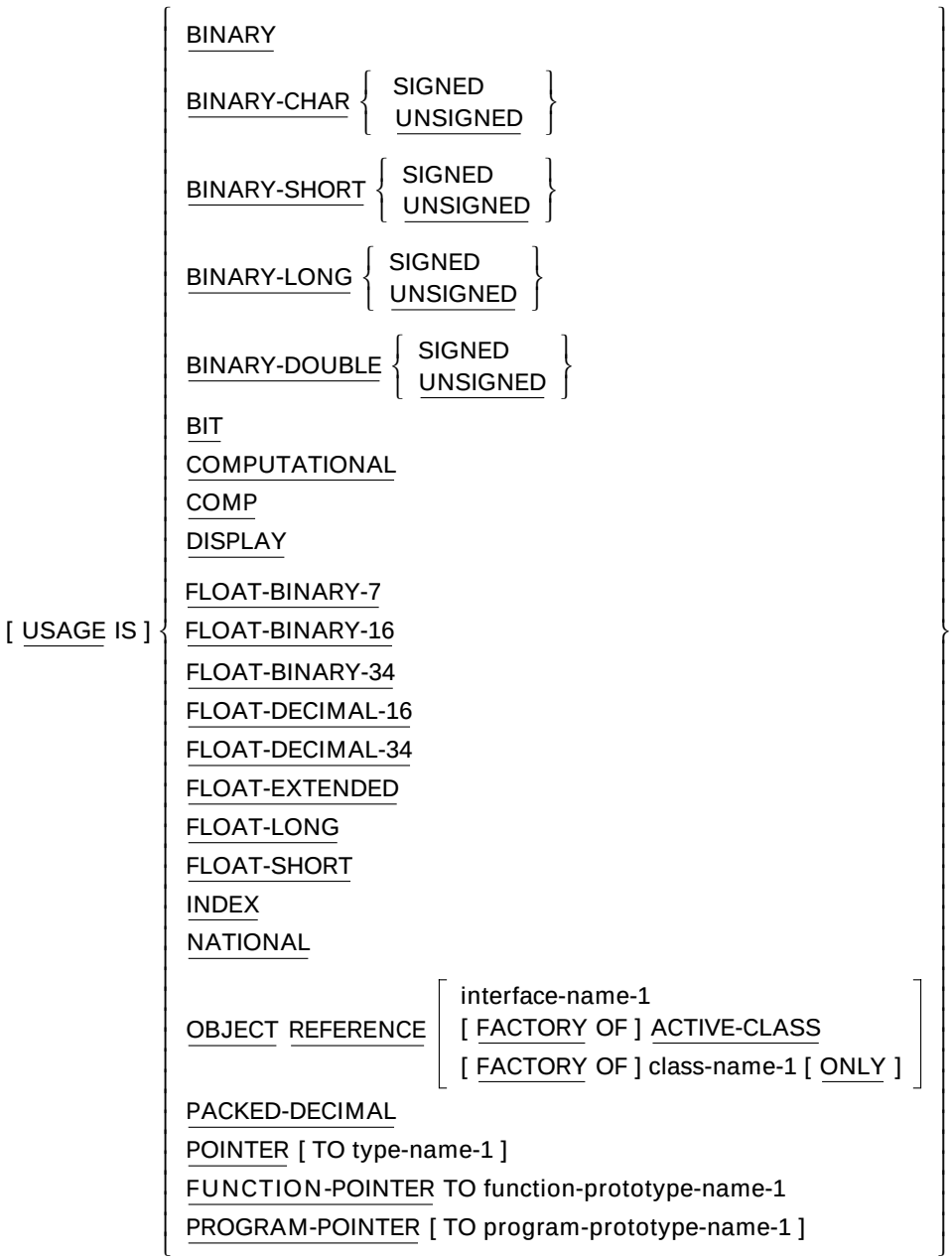
1) If the UNDERLINE clause is specified at group level, it applies to each elementary screen item in that group.

2) When the UNDERLINE clause is specified, the screen item will be displayed so that the characters that constitute the screen item are underlined when the screen item is referenced in an ACCEPT screen or a DISPLAY screen statement.

13.18.59 USAGE clause

The USAGE clause specifies the representation of a data item in the computer storage.

13.18.59.1 General format



13.18.59.2 Syntax rules

1) The USAGE clause shall not be specified in a data description entry that has a level-number of 66 or 88.

2) If the USAGE clause is written in the data description entry for a group item, it may also be written in the data description entry for any subordinate elementary item or group item, but the same usage shall be specified in both entries.

3) An elementary data item whose declaration contains, or an elementary data item subordinate to a group item whose declaration contains, a USAGE clause specifying BINARY, COMPUTATIONAL, or PACKED-DECIMAL shall be specified only with a picture character-string that describes a numeric item.

4) The INDEX, OBJECT REFERENCE, POINTER, FUNCTION-POINTER, and PROGRAM-POINTER phrases shall not be specified in a data item described with the CONSTANT RECORD clause, or in any item subordinate to a data item described with the CONSTANT RECORD clause.

5) An elementary data item with usage bit shall be specified only with a picture character-string that describes a boolean data item.

6) COMP is an abbreviation for COMPUTATIONAL.

7) Only the DISPLAY or NATIONAL phrase may be specified in any USAGE clause associated with a report group item.

8) A program-pointer data item may be referenced explicitly only in a CALL statement, an INITIALIZE statement, an INVOKE statement, a SET statement, a relation condition, a procedure division header, the argument list of an inline invocation of a method, and as an argument in a function-identifier.

9) A data-pointer data item may be referenced explicitly only in a CALL statement, an INITIALIZE statement, an INVOKE statement, a SET statement, a relation condition, a procedure division header, the argument list of an inline invocation of a method, as an argument in a function-identifier, in the RETURNING phrase of an ALLOCATE statement, or in a FREE statement.

10) An index data item may be referenced explicitly only in a SEARCH or SET statement, a relation condition, an intrinsic function argument, an inline method invocation argument, the USING phrase of a procedure division header, or the USING phrase of a CALL or INVOKE statement.

11) An elementary data item of class index, object, or pointer shall not be a conditional variable.

12) An elementary data item with usage national shall be declared with a picture character-string that describes a boolean, national, national-edited, numeric, or numeric-edited data item.

13) When a USAGE clause is not specified for an elementary data item or for any group to which the data item belongs:

    a) if the explicit or implicit picture character-string contains the symbol 'N', a USAGE NATIONAL clause is implied;

    b) otherwise, a USAGE DISPLAY clause is implied.

14) A USAGE clause with the OBJECT REFERENCE, POINTER, FUNCTION-POINTER, or PROGRAM-POINTER phrase may be specified only for an elementary data item with level 1 or subordinate to a type declaration that includes the STRONG phrase.

15) The USAGE OBJECT REFERENCE clause shall not be specified in the file section.

16) The ACTIVE-CLASS phrase may be specified only in a factory definition, an instance definition, or the linkage or local-storage section of a method definition.

17) In a screen description entry, only DISPLAY or NATIONAL may be specified.

18) If type-name-1 is specified, the TYPEDEF clause shall be specified for the subject of the entry.

19) If program-prototype-name-1 is specified, the TYPEDEF clause shall be specified for the subject of the entry.

20) Only the NATIONAL phrase may be specified in a USAGE clause associated with an elementary data item whose explicit or implicit picture character-string contains the symbol 'N'.

13.18.59.3 General rules

1) If the USAGE clause is specified or implied at a group level, it applies only to each elementary item in the group. Unless the GROUP-USAGE clause is also specified or implied, the USAGE clause applies only to each elementary item in the group and not to the group itself.

2) The USAGE clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the procedure division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

3) The VALIDATE statement may be used to check that the content of a data item is compatible with any PICTURE, SIGN and USAGE clauses specified in the description of the item. Rules for compatibility are implementor-defined. Failure of this check results in the setting of the data item's internal indicator to invalid on format. A data item with usage index, object reference, pointer, function-pointer, or program-pointer is ignored by the VALIDATE statement.

4) The USAGE BINARY clause specifies that a radix of 2 is used to represent a numeric item in the storage of the computer. Each implementor specifies the precise effect of the USAGE BINARY clause upon the alignment and representation of the data item in the storage of the computer, including the representation of any algebraic sign. Sufficient computer storage shall be allocated by the implementor to contain the maximum range of values implied by the associated decimal picture character-string.

5) The USAGE BIT clause specifies that bits shall be used to represent a boolean data item. A data item described with USAGE BIT is a bit data item. The alignment of a data item described with USAGE BIT is specified in 8.5.1.5.3, Alignment of data items of usage bit.

6) The USAGE COMPUTATIONAL clause specifies that a radix and format specified by the implementor is used to represent a numeric item in the storage of the computer. Each implementor specifies the precise effect of the USAGE COMPUTATIONAL clause upon the alignment and representation of the data item in the storage of the computer, including the representation of any algebraic sign, and upon the range of values that the data item may hold.

7) The implicit or explicit USAGE DISPLAY clause specifies that an alphanumeric coded character set shall be used to represent a data item in the storage of the computer, and that the data item is aligned on a character boundary. Alphanumeric characters shall be represented in the storage of the computer as characters of uniform size equal to or less than the size of characters in the computer's national character set. Each implementor shall specify the size and representation of characters stored for usage DISPLAY.

8) The implicit or explicit USAGE NATIONAL clause specifies that a national coded character set shall be used to represent a data item in the storage of the computer, and that the data item shall be aligned on a character boundary. National characters shall be represented in the storage of the computer as characters of a uniform size equal to or a multiple of the size of characters in the computer's alphanumeric character set. Each implementor shall specify the size and representation of characters stored for usage NATIONAL.

9) The USAGE INDEX clause specifies that a data item is an index data item and contains a value that shall correspond to an occurrence number of a table element. The class and category of an index data item are index. Each implementor specifies the precise effect of the USAGE INDEX clause upon the alignment and representation of the data item in the storage of the computer, including the actual value assigned for any given occurrence number.

10) The USAGE PACKED-DECIMAL clause specifies that a radix of 10 is used to represent a numeric item in the storage of the computer. Furthermore, this clause specifies that each digit position shall occupy the minimum possible configuration in computer storage. Each implementor specifies the precise effect of the USAGE PACKED-DECIMAL clause upon the alignment and representation of the data item in the storage of the

computer, including the representation of any algebraic sign. Sufficient computer storage shall be allocated by the implementor to contain the maximum range of values implied by the associated decimal picture character-string.

11) The usages binary-char, binary-short, binary-long, and binary-double define integer numeric data items that are held in a binary format suitable for the machine on which the runtime module is to run. The minimum range requirements for each of these usages are shown below. Any integer value of n within the specified range shall be expressible in the given usage. The implementor may allow a wider range. However, for signed items, any number that may be expressed as BINARY-CHAR shall also be expressible as BINARY-SHORT, any number that may be expressed as BINARY-SHORT shall also be expressible as BINARY-LONG, and any number that may be expressed as BINARY-LONG shall also be expressible as BINARY-DOUBLE.

    The minimum ranges are:

BINARY-CHAR SIGNED	-128 [-2**7]	<	n	<	128 [2**7]
BINARY-CHAR UNSIGNED	0	≤	n	<	256 [2**8]
BINARY-SHORT SIGNED	-32768 [-2**15]	<	n	<	32768 [2**15]
BINARY-SHORT UNSIGNED	0	≤	n	<	65536 [2**16]
BINARY-LONG SIGNED	-2147483648 [-2**31]	<	n	<	2147483648 [2**31]
BINARY-LONG UNSIGNED	0	≤	n	<	4294967296 [2**32]
BINARY-DOUBLE SIGNED	-2**63	<	n	<	2**63
BINARY-DOUBLE UNSIGNED	0	≤	n	<	2**64

12) The usages float-short, float-long and float-extended define signed numeric data items that are held in a floating-point format suitable for the machine on which the runtime module is to run. The size and permitted range of values for these fields is defined by the implementor. Any value that may be held in a data item of usage float-short shall also be expressible in a data item of usage float-long. Any value that may be held in a data item of usage float-long shall also be expressible in a data item of usage float-extended.

13) The FLOAT-BINARY-7 phrase specifies that the data item is a floating-point data item in the format specified as the 32-bit binary interchange floating-point format (‘Binary32’) in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings and Table 3.5, Binary interchange format parameters.

14) The FLOAT-BINARY-16 phrase specifies that the data item is a floating-point data item in the format specified as the 64-bit binary interchange floating-point format (‘Binary64’) in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings and Table 3.5, Binary interchange format parameters.

15) The FLOAT-BINARY-34 phrase specifies that the data item is a floating-point data item in the format specified as the 128-bit binary interchange floating-point format (‘Binary128’) in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings and Table 3.5, Binary interchange format parameters.

16) The FLOAT-DECIMAL-16 phrase specifies that the data item is a floating-point data item in the format specified as the 64-bit decimal interchange floating-point format (‘Decimal64’) using the decimal encoding as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.5, Decimal interchange format encodings, limited to decimal encodings for the significand, and Table 3.6, Decimal interchange format parameters.

17) The FLOAT-DECIMAL-34 phrase specifies that the data item is a floating-point data item in the format specified as the 128-bit decimal interchange floating-point format (‘Decimal128’) using the decimal encoding as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.5, Decimal interchange format encodings, limited to decimal encodings for the significand, and Table 3.6, Decimal interchange format parameters.

18) The representation and length of a data item described with USAGE BINARY-CHAR, BINARY-SHORT, BINARY-LONG, BINARY-DOUBLE, FLOAT-SHORT, FLOAT-LONG, or FLOAT-EXTENDED is implementor-defined. The length and alignment of a data item described with the SIGNED phrase shall be the same as the length and alignment of a data item described with the UNSIGNED phrase.

19) A data item described with a USAGE OBJECT REFERENCE clause is called an object reference. An object reference is a data item of class object and category object-reference. It shall contain either null or a reference to an object, subject to the following rules:

    a) The amount of storage allocated for an object reference data item is implementor-defined and is not necessarily the same for every object reference.

    b) If none of the optional phrases is specified, this data item is called a universal object reference. Its content may be a reference to any object.

    c) If interface-name-1 is specified, the object referenced by this data item shall implement interface-1.

    d) If class-name-1 is specified, the object referenced by this data item shall be an object of class-name-1 or of a subclass of class-name-1, subject to the following rules:

        1. If the ONLY phrase is not specified:

            a. If the FACTORY phrase is specified, the object referenced by this data item shall be the factory object of the specified class or of a subclass of the specified class.

            b. If the FACTORY phrase is not specified, the object referenced by this data item shall be an instance object of the specified class or of a subclass of the specified class.

        2. If the ONLY phrase is specified:

            a. If the FACTORY phrase is specified, the object referenced by this data item shall be the factory object of the specified class.

            b. If the FACTORY phrase is not specified, the object referenced by this data item shall be an instance object of the specified class.

    e) If ACTIVE-CLASS is specified, the object referenced by this data item shall be of the same class as the object that was used to invoke the method in which this data description entry is specified, subject to the following rules:

        1. If the FACTORY phrase is specified, the object referenced by this data item shall be the factory object of that class.

        2. If the FACTORY phrase is not specified, the object referenced by this data item shall be an instance object of that class.

20) A data description entry that specifies the USAGE POINTER clause specifies a data-pointer data item, also called a data-pointer, that may contain the address of a data item. Each implementor defines the precise effect of the USAGE POINTER clause upon alignment, size, representation, and range of values of the data item in the storage of the computer. If type-name-1 is specified, this data item is a restricted data-pointer. A restricted data-pointer shall contain only the predefined address NULL or the address of a data item of the specified type.

21) A data description entry that specifies the USAGE PROGRAM-POINTER clause specifies a program-pointer data item, also called a program-pointer, that may contain the address of a program. Each implementor defines the precise effect of the USAGE PROGRAM-POINTER clause upon alignment, size, and representation of the data item in the storage of the computer.

The program addressed by a program-pointer data item may be written in COBOL or in another language for which the implementor has declared support for access by program pointers. For a COBOL program, the address is for an outermost program.

22) If program-prototype-name-1 is specified, this data item is a restricted program-pointer. A restricted program-pointer shall contain only the predefined address NULL or the address of a program with the same signature as that identified by the specified program-prototype-name.

23) A data description entry that specifies the USAGE FUNCTION-POINTER clause specifies a function-pointer data item, also called a function-pointer, that may contain the address of a function. Each implementor defines the precise effect of the USAGE FUNCTION-POINTER clause upon alignment, size, and representation of the data item in the storage of the computer.

The function addressed by a function-pointer data item may be written in COBOL or in another language for which the implementor has declared support for access by function pointers.

A function-pointer shall contain only the predefined address NULL or the address of a function with the same signature as that identified by the specified function-prototype-name-1.

13.18.60 USING clause

The USING clause identifies data to be used both as the destination in an ACCEPT screen statement and the source for a DISPLAY screen statement.

13.18.60.1 General format

USING identifier-1

13.18.60.2 Syntax rules

1) The category of the data item referenced by identifier-1 shall be a permissible category as a receiving operand in a MOVE statement where the sending operand has the same PICTURE clause as the subject of the entry.

2) The category of the data item referenced by identifier-1 shall be a permissible category as a sending operand in a MOVE statement where the receiving operand has the same PICTURE clause as the subject of the entry.

3) Identifier-1 shall be defined in the file, working-storage, local-storage, or linkage section. Identifier-1 shall not specify a variable-length group.

4) If the subject of this entry is subject to an OCCURS clause, identifier-1 shall be specified without the subscripting normally required. Additional requirements are specified in 13.18.37, OCCURS clause, syntax rule 13.

13.18.60.3 General rules

1) Specifying the USING clause is equivalent to specifying both the TO and FROM clauses, each specifying the same identifier.

2) The subject of the entry is both an input screen item and an output screen item.

13.18.61 VALIDATE-STATUS clause

The VALIDATE-STATUS clause enables messages and flags to be generated automatically when a specific error condition arises or does not arise as a result of the execution of a VALIDATE statement.

13.18.61.1 General format

```
{ VALIDATE-STATUS } IS { identifier-1 } WHEN { ERROR    } [ ON { FORMAT   } ]
{ VAL-STATUS     }    { literal-1    }      { NO ERROR }      { CONTENT  }
                                                             { RELATION }
                FOR { identifier-2 } ...
```

13.18.61.2 Syntax rules

1) Literal-1 or the data item referenced by identifier-1 shall be valid as a sending operand for a MOVE statement specifying the subject of the entry as the receiving operand.

2) Literal-1 shall not be a zero-length literal.

3) The entry containing the VALIDATE-STATUS clause shall not contain or be subordinate to an OCCURS clause with the DEPENDING phrase.

4) The data item referenced by identifier-2 shall be the operand of a VALIDATE statement, or shall be subordinate to such an item.

5) Identifier-2 shall not be reference modified.

6) Identifier-2 shall not be of class index, object, or pointer.

7) If identifier-2 references a data item whose data description entry is not subject to any OCCURS clause, the VALIDATE-STATUS clause shall not be specified in an entry that is subject to any OCCURS clause.

   If identifier-2 references a data item whose data description entry is subject to one or more OCCURS clauses, the following rules apply:

   a) If the VALIDATE-STATUS clause is specified in an entry that is not subject to any OCCURS clause, identifier-2 shall be subscripted. The number of subscripts shall be the same as the number of nested OCCURS clauses to which the data item referenced by identifier-2 is subject. Only literal subscripts may be specified.

   b) If the VALIDATE-STATUS clause is specified in an entry that is subject to one or more OCCURS clauses, identifier-2 shall not be subscripted. The data item referenced by identifier-2 and the subject of the entry shall both be subject to the same number of OCCURS clauses and the number of occurrences specified in the corresponding OCCURS clauses shall be equal. If an OCCURS clause associated with identifier-2 has a DEPENDING phrase, the maximum integer of the TO phrase is used in determining the number of occurrences of identifier-2.

8) If identifier-2 references a global name, the subject of the entry shall be global.

9) The words VAL-STATUS and VALIDATE-STATUS are equivalent.

13.18.61.3 General rules

1) The VALIDATE-STATUS clause takes effect during the final, error indication stage of the execution of a VALIDATE statement that directly or indirectly references the data item referenced by identifier-2. The VALIDATE-STATUS clause is ignored during the execution of any statement other than a VALIDATE statement.

2) The ERROR phrase takes effect in the following circumstances:

   a) if ON phrase is not specified, when the associated internal indicator of the data item referenced by identifier-2 is not set to its initial valid value;

   b) if the ON phrase is specified, when the associated internal indicator of the item referenced by identifier-2 is set to the value or one of the values corresponding to the stage of validation specified in the ON phrase.

3) The NO ERROR phrase takes effect in the following circumstances:

   a) if ON phrase is not specified, when the associated internal indicator of the data item referenced by identifier-2 is still set to its initial valid value;

   b) if the ON phrase is specified, when the associated internal indicator of the item referenced by identifier-2 is not set to the value or one of the values corresponding to the stage of validation specified in the ON phrase.

4) The ERROR and NO ERROR phrases take effect by executing an implicit MOVE statement in which literal-1 or the current content of the data item referenced by identifier-1 is the sending operand and the data item that is the subject of the entry is the receiving operand.

5) If a data description entry contains more than one VALIDATE-STATUS clause, these clauses are applied as described above in the order in which they are specified until either the action defined under general rule 4 has occurred or all the clauses have been processed.

6) If neither an ERROR phrase nor a NO ERROR phrase for a given entry takes effect during the execution of a given VALIDATE statement, the effect on the data item that is the subject of the entry is as follows: if any VALIDATE-STATUS clause specified in the entry has an identifier-2 that is referenced directly or indirectly as the operand of the VALIDATE statement, the data item is initialized by the execution of an implicit INITIALIZE statement without the VALUE or REPLACING phrases; otherwise, the data item remains unchanged.

7) Data items that are subordinate to the operand of a VALIDATE statement and are not processed during the statement's execution, as indicated by a value in their internal indicators signifying not processed, are considered to be neither valid nor invalid by the VALIDATE-STATUS clause but will cause the VALIDATE-STATUS clause's data item to be initialized, where applicable, as described in general rule 6. (See 13.18.37, OCCURS clause, and 13.18.40, PRESENT WHEN clause.)

8) If the entry containing the VALIDATE-STATUS clause is subject to one or more OCCURS clauses, the processing defined in the above general rules applies to each occurrence of the subject of the entry and each corresponding occurrence of the data item referenced by identifier-2.

13.18.62 VALUE clause

The VALUE clause specifies the initial value of local-storage section and working-storage section data items, the values used by explicit and implicit INITIALIZE statements, the values associated with condition-names, the value of report section printable items and screen section displayable items, and the values used by the VALIDATE statement.

13.18.62.1 General format

Format 1 (data-item):

```
VALUE IS literal-1
```

Format 2 (table):

```
{ VALUE IS  }
{ VALUES ARE } { { literal-1 } ... FROM ( { subscript-1 } ... ) [ TO ( { subscript-2 } ... ) ] } ...
```

Format 3 (condition-name):

```
{ VALUE IS  }   { literal-2 [ { THROUGH } literal-3 ] }   ... [ IN alphabet-name-1 ]
{ VALUES ARE }                { { THRU    }           }
                [ WHEN SET TO FALSE IS literal-4 ]
```

Format 4 (report-section):

```
{ VALUE IS  }
{ VALUES ARE } { literal-1 } ...
```

Format 5 (content-validation-entry):

```
{ VALUE  }   { literal-5 [ { THROUGH } literal-6 ] }   ... [ IN alphabet-name-1 ]
{ VALUES }                 { THRU    }             }

[ IS  ] { INVALID } [ WHEN condition-1 ]
[ ARE ] { VALID   }
```

13.18.62.2 Syntax rules

ALL FORMATS

1) The subject of the entry shall not be a strongly-typed group item or a variable-length group.

2) If the category of the subject of the entry is numeric, all literals in the VALUE clause shall be numeric and have a value that is within the range of values indicated by the PICTURE clause or the USAGE clause. If the usage is other than float-short, float-long and float-extended, this value shall not require truncation of nonzero digits. This value shall have the digit zero in each position that corresponds to a picture character symbol 'P'.

3) If a signed numeric literal is specified, the subject of the entry shall be a signed numeric data item.

4) If the item is of class alphabetic or alphanumeric, literals in the VALUE clause shall be alphanumeric literals. Alphanumeric literals in the VALUE clause of an elementary item shall not exceed the size indicated by an explicit PICTURE clause. Alphanumeric literals in the VALUE clause of an alphanumeric group item shall not exceed the size of the group item.

5) If the item is of class national, literals in the VALUE clause shall be national literals. National literals in the VALUE clause of an elementary item shall not exceed the size indicated by an explicit PICTURE clause. National literals in the VALUE clause of a national group item shall not exceed the size of the group item.

FORMATS 1, 2, 3, AND 5

6) If the category of the subject of the entry is boolean, literals in the VALUE clause shall be boolean literals. Boolean literals in the VALUE clause of an elementary item shall not exceed the size indicated by an explicit PICTURE clause. Boolean literals in the VALUE clause of a bit group item shall not exceed the size of the group item.

FORMATS 2, 3, 4, AND 5

7) The words VALUE and VALUES are equivalent.

FORMATS 1, 2, AND 3

8) Editing characters in a picture character-string are included in determining the size of the data item, but do not cause editing of the initial value when the data item is initialized.

NOTE The programmer is responsible for specifying the value of a literal associated with an alphanumeric-edited, numeric-edited, or national-edited item in edited form.

FORMATS 3, 4, and 5

9) A format 3, 4, or 5 VALUE clause shall not be specified in any data description entry that contains the CONSTANT RECORD clause, or in any data description entry subordinate to a data description entry that contains the CONSTANT RECORD clause.

FORMATS 1 AND 2

10) Formats 1 and 2 shall not be specified in a condition-name entry.

11) The VALUE clause shall not be specified in a data description entry that contains a REDEFINES clause or in an entry that is subordinate to an entry containing a REDEFINES clause.

12) If the VALUE clause is specified at the group level, literal-1 shall be of the same category as the group item or shall be a figurative constant that is permitted in a MOVE statement to a receiving item of that category. The VALUE clause shall not be specified at subordinate levels within this group.

13) If a VALUE clause is specified at the group level, subordinate items within that group shall not be described with a JUSTIFIED or SYNCHRONIZED clause, and all data items subordinate to an alphanumeric group item shall be explicitly or implicitly described with usage DISPLAY.

FORMAT 1

14) In the screen section, literal-1 shall be an alphanumeric literal if usage display is specified or a national literal if usage national is specified. Literal-1 shall not be a figurative constant.

FORMAT 2

15) A data description entry that contains the VALUE clause shall contain an OCCURS clause or be subordinate to a data description entry that contains an OCCURS clause.

16) Subscript-1 and subscript-2 shall be integer numeric literals.

17) In one FROM phrase, there shall be one subscript-1 specified for each OCCURS clause for the subject of the entry or superordinate to that entry, specified in the same order as a subscripted reference to the subject of the entry would be specified. Each instance of subscript-1 shall not exceed the maximum number of occurrences specified in the OCCURS clause associated with that instance.

18) If the TO phrase is specified, there shall be one subscript-2 specified for each OCCURS clause for the subject of the entry or superordinate to that entry, specified in the same order as a subscripted reference to the subject of the entry would be specified. Each instance of subscript-2 shall not exceed the maximum number of occurrences specified in the OCCURS clause associated with that instance. The specification of subscript-2 in one TO phrase shall be such that the table element associated with subscript-2 is the same occurrence or a successive occurrence of the table element associated with the corresponding subscript-1.

19) A VALUE clause without the TO phrase shall not be specified in the same entry as an OCCURS clause with a DYNAMIC phrase but no TO phrase, or in any entry subordinate to such an OCCURS clause.

20) If the TO phrase is specified and an OCCURS clause with a DYNAMIC phrase but no TO phrase is specified in the same entry or in any superordinate entry, the values of subscript-1 and subscript-2 corresponding to all levels higher than that of the OCCURS clause, if applicable, shall be equal.

FORMAT 3

21) When the THROUGH phrase is specified:

    a) when literal-2 is of a class other than alphanumeric or national, the value of literal-2 shall be less than the value of literal-3.

    b) when literal-2 is of class alphanumeric or national, and the runtime collating sequence is known, the value of literal-2 shall be less than the value of literal-3. The runtime collating sequence is unknown when the collating sequence is defined by a locale or the collating sequence is otherwise determined at runtime.

        NOTE The compiler may know the runtime collating sequence for a variety of reasons; perhaps only one is possible or one is specified by a compiler option. However, this is not a requirement of an implementor, and specifying an alphabet-name for a fixed collating sequence ensures that the range of values will always be the same.

22) The value of literal-4 shall not be equal to the value of any occurrence of literal-2. When the THROUGH phrase is specified:

    a) when literal-2 is of a class other than alphanumeric or national, the value of literal-4 shall not be equal to any value in the range of any occurrence of literal-2 through literal-3, inclusive.

    b) when literal-2 is of class alphanumeric or national, and the runtime collating sequence is known, the value of literal-4 shall not be equal to the value of any literal-2 or any value in the range of any occurrence of literal-2 through literal-3, inclusive.

        NOTE When the THROUGH phrase is specified and the runtime collating sequence is not known at compile time, it is possible that the value used in SET condition-name TO FALSE is also a value for which the condition will evaluate TRUE. This can be avoided by specifying a list of values rather than specifying a range in a THROUGH phrase.

FORMATS 3 AND 5

23) The words THROUGH and THRU are equivalent.

24) If the category of the subject of the entry is boolean, the THROUGH phrase shall not be specified.

25) Condition-name and content-validation formats shall not be specified in the report section.

26) Alphabet-name-1 may be specified only when the literals specified in the THROUGH phrase are of class alphanumeric or national. If literal-2 and literal-3 or literal-5 and literal-6 are of class national, alphabet-name-1 shall reference an alphabet that defines a national collating sequence; otherwise, alphabet-name-1 shall reference an alphabet that defines an alphanumeric collating sequence.

27) The manner of determining the range of values specified by the THROUGH phrase is described in 14.7.7, THROUGH phrase.

28) Formats 3 and 5 may be specified only when the level-number of the subject of the entry is 88.

FORMAT 4

29) If the VALUE clause has more than one operand, the entry shall be a repeating entry or shall be subordinate to a repeating entry. The number of operands of the VALUE clause shall be equal to the number of repetitions of the repeating entry or the same number multiplied by the number of repetitions of any number of successive repeating entries at higher levels than the repeating entry.

FORMAT 5

30) The VALID phrase and the INVALID phrase shall not both be specified for the same conditional variable.

31) When the THROUGH phrase is specified:

    a) when literal-6 is of a class other than alphanumeric or national, the value of literal-5 shall be less than the value of literal-6.

    b) when literal-5 is of class alphanumeric or national, and the runtime collating sequence is known, the value of literal-5 shall be less than the value of literal-6. The runtime collating sequence is unknown when the collating sequence is defined by a locale or the collating sequence is otherwise determined at runtime.

        NOTE The compiler may know the runtime collating sequence for a variety of reasons; perhaps only one is possible or one is specified by a compiler option. However, this is not a requirement of an implementor, and specifying an alphabet-name for a fixed collating sequence ensures that the range of values will always be the same.

32) If the word VALUE is specified, the word IS may be specified. If the word VALUES is specified, the word ARE may be specified.

13.18.62.3 General rules

FORMATS 1, 2, 3, AND 4

1) If the usage of the subject of the entry is float-short, float-long or float-extended, the actual value given to the item is an approximation of the arithmetic value of the literal.

FORMATS 1 AND 2

2) In the file section, VALUE clauses take effect only during the execution of an INITIALIZE statement. The initial value of the data items in the file section is undefined.

3) In the linkage section, VALUE clauses take effect only during the execution of an explicit or implicit INITIALIZE statement. The initial value of the data items in the linkage section is specified in 13.7, Linkage section.

4) VALUE clauses in the working-storage and local-storage sections take effect:

a) for external items, during the execution of an INITIALIZE statement

b) for based items and their subordinate items, during the execution of an ALLOCATE or an explicit or implicit INITIALIZE statement

c) for all other items

— when the object or runtime element is placed in initial state, and
— during the execution of an INITIALIZE statement.

When VALUE clauses take effect, data items with a VALUE clause are initialized to the specified value and data items of class object and class pointer are initialized to null; the values of other data items and indexes are undefined.

5) If a VALUE clause is specified in a data description entry of a group item, the group area is initialized without consideration for the individual elementary or group items contained within this group.

6) If a VALUE clause is specified in a data description that also contains an OCCURS clause with the DEPENDING phrase, the initialization of the associated data item behaves as if the value of the data item referenced by the DEPENDING phrase in the OCCURS clause specified for the variable-occurrence data item is set to the maximum number of occurrences as specified by that OCCURS clause. A data item is associated with an occurs-depending table when a data item is one of the following:

a) a group data item that contains an occurs-depending table.

b) an occurs-depending table.

c) a data item that is subordinate to an occurs-depending table.

If a VALUE clause is associated with the data item referenced by a DEPENDING phrase, that value is considered to be placed in the data item after the occurs-depending table is initialized. Further rules for initializing tables are specified in 13.18.37, OCCURS clause.

FORMATS 1, 2, AND 4

7) Each literal is aligned in the associated data item in accordance with 14.6.8, Alignment of data within data items, except that initialization is not affected by a JUSTIFIED clause and no editing takes place.

8) Initialization is not affected by any BLANK WHEN ZERO clause that may be specified.

FORMAT 1 AND 4

9) A VALUE clause specified in a data description entry that contains an OCCURS clause or in an entry that is subordinate to an OCCURS clause causes every occurrence of the associated data item to be assigned the specified value.

FORMATS 3 AND 5

10) The manner of determining the range of values specified by the THROUGH phrase is described in 14.7.7, THROUGH phrase.

NOTE The range of values can be determined at compile time only when the runtime collating sequence is known at compile time.

FORMAT 1

11) If a VALUE clause is specified for a screen item, literal-1 specifies the value of the screen item that is displayed on the screen when directly or indirectly referenced by an ACCEPT screen statement or a DISPLAY screen statement.

FORMAT 2

12) A format 2 VALUE clause initializes a table element to the value of literal-1. The table element initialized is identified by subscript-1. Consecutive table elements are initialized, in turn, to the value of successive occurrences of the literal-1. Consecutive table elements are referenced by incrementing by 1 the subscript that represents the least inclusive dimension of the table. When any reference to a subscript, prior to incrementing it, is equal to the maximum number of occurrences, or, in the case of a dynamic-capacity table, the expected number of occurrences, specified by its corresponding OCCURS clause, that subscript is set to 1 and the subscript for the next most inclusive dimension of the table is incremented by 1.

13) If the TO phrase is specified, all occurrences of literal-1 are reused, in the order specified, as a source during the initialization described in general rule 12. This repetition is done until the table element identified by subscript-2 is initialized.

14) If the TO phrase is not specified, it is as if the TO phrase were specified with each subscript-2 as the maximum number of occurrences, or, in the case of a dynamic-capacity table, the expected number of occurrences, of the table associated with each corresponding subscript-1.

15) If multiple specifications of the FROM phrase reference the same table element, the value defined by the last specified FROM phrase in the VALUE clause is assigned to the table element.

16) If an OCCURS clause with the DYNAMIC phrase is specified in the same entry as the VALUE clause, or in any entry superordinate to it, the initial capacity of the associated dynamic-capacity table is calculated according to the following subrules. If more than one VALUE clause applies, the maximum value thus calculated becomes the initial capacity.

    a) If a TO phrase is specified in the VALUE clause, the initial capacity is increased, if necessary, to the value of the corresponding subscript-2, provided that this value does not lie outside the range defined by the minimum and expected capacity specified in the OCCURS clause. If the value of subscript-2 lies outside this range, the initial capacity is unchanged.

    b) If no TO phrase is specified in the VALUE clause, the initial capacity is set equal to the expected capacity specified in the OCCURS clause.

FORMAT 3

17) The characteristics of a condition-name are implicitly those of its conditional variable.

18) When a condition-name is referenced in a 'SET condition-name TO FALSE' statement, the value of literal-4 from the FALSE phrase is placed in the associated conditional-variable.

    NOTE The true values of a conditional-variable are all the values associated with its condition-names; all other values are false values. The WHEN SET TO FALSE phrase specifies just one of possibly many false values; the purpose is to define a single value to be used by 'SET condition-name TO FALSE'. When a condition-name condition is tested, all the non-true values give a false result, not just the one false value defined by the WHEN SET TO FALSE phrase.

FORMAT 4

19) The value of literal-1 is used for the content of the printable item whenever it is printed, except that a GROUP INDICATE, PRESENT WHEN, or OCCURS clause with the DEPENDING phrase may suppress the appearance of the item, as described in 13.18.27, GROUP INDICATE clause; 13.18.40, PRESENT WHEN clause; and 13.18.37, OCCURS clause.

20) If the VALUE clause has more than one operand, successive operands are assigned to successive repeating printable items, horizontally and then vertically, as applicable, in that hierarchy. If no further operands remain, assignment begins again from the first operand. If any of the printable items are suppressed as a result of a PRESENT WHEN clause or an OCCURS clause with a DEPENDING phrase, VALUE operands are nevertheless assigned to them, even though they are not printed.

FORMAT 5

21) Format 5 entries take effect during the content validation stage of the execution of a VALIDATE statement.

22) If the associated conditional variable is an elementary item, the content validation entries take effect only if the conditional variable's associated internal indicator indicates that the item is valid on format.

23) If the WHEN phrase is specified, condition-1 is evaluated. If condition-1 is true, the clause takes effect as described in the following general rules; otherwise, that content validation entry is ignored for the purpose of the current VALIDATE statement and the effect is as though that content validation entry had been omitted.

24) Each entry with the VALID phrase defines a set of allowable values or ranges of values for the associated data item. If several entries with the VALID phrase are associated with the same data item, the union of all their sets of values defines the allowable set of values. If this check fails, the item's associated internal indicator is set to invalid on content.

25) Each entry with the INVALID phrase defines a set of forbidden values or ranges of values for the associated data item. If several entries with the INVALID phrase are associated with the same data item, the union of all their sets of values defines the forbidden set of values. If this check fails, the item's associated internal indicator is set to invalid on content.

13.18.63 VARYING clause

The VARYING clause establishes counters to enable different source items to be placed in each occurrence of a repeated printable item in report writer.

The VARYING clause also establishes counters to enable a VALIDATE statement to store different occurrences of a data item in different occurrences of a destination data item, or to compare different occurrences of a data item during content or relation validation.

13.18.63.1 General format

VARYING { data-name-1 [FROM arithmetic-expression-1] [BY arithmetic-expression-2] } ...

13.18.63.2 Syntax rules

1) The entry containing the VARYING clause shall also contain an OCCURS clause or, if the VARYING clause appears in a report group description entry, a multiple LINE or multiple COLUMN clause.

2) Data-name-1 shall not be defined elsewhere in the source element, except as data-name-1 in another VARYING clause of an entry not subordinate to the subject of the current entry.

   NOTE Such a reuse refers to a completely independent data item.

   This definition of data-name-1 may be referenced only within the current entry or a subordinate entry.

3) Data-name-1 shall not be referenced in arithmetic-expression-1 of the same VARYING clause, but may be referenced in arithmetic-expression-2 of the same VARYING clause or in arithmetic-expression-1 or arithmetic-expression-2 of a VARYING clause in a subordinate entry.

13.18.63.3 General rules

1) Each entry containing a VARYING clause establishes an independent temporary integer data item that shall be large enough to contain the maximum expected value.

2) If the VARYING clause is not specified in a report description entry, it is ignored during the execution of any statement other than a VALIDATE statement.

3) When the data item that is the subject of the entry is processed by the VALIDATE statement or the report item that is the subject of the entry is processed, the content of data-name-1 is established for each repetition of the associated data item during each stage of execution of the VALIDATE statement or each repetition of the report item, as follows:

   a) For the first occurrence, the value of arithmetic-expression-1 is moved to data-name-1. If the FROM phrase is absent, 1 is moved to data-name-1.

   b) For the second and subsequent occurrences, the value of arithmetic-expression-2 is added to data-name-1. If the BY phrase is absent, 1 is added to data-name-1.

4) Each value of data-name-1, established in this way, persists throughout the processing of the associated occurrence of the data item or report item.

   NOTE For example, this allows data-name-1 to be used as a source data item, as a subscript to a source data item or as part of the identifier in the DEFAULT clause.

5) If the evaluation of arithmetic-expression-1 or arithmetic-expression-2 produces a noninteger value and the VARYING clause was specified in a report description entry, the EC-REPORT-VARYING exception condition is set to exist, the execution of the GENERATE statement is unsuccessful, and the content of the print line is undefined.

6) If the evaluation of arithmetic-expression-1 or arithmetic-expression-2 produces a noninteger value and the VARYING clause was not specified in a report description entry, the EC-VALIDATE-VARYING exception condition is set to exist, the execution of the VALIDATE statement is unsuccessful, and the content of the receiving items is undefined.

14 Procedure division

14.1 General

The procedure division in a function, a method, or a program contains procedures to be executed.

The procedure division in an instance definition and a factory definition contains the methods that may be invoked on the instance object or factory object.

The procedure division in an interface contains method prototypes.

The procedure division in a function prototype, method prototype, or program prototype specifies the parameters, any returning item, and any exceptions that may be raised.

14.2 Procedure division structure

14.2.1 General format

Format 1 (with-sections):

```
procedure-division-header
[ DECLARATIVES.
{ section-name-1 SECTION.
     use-statement.
  [ sentence ] ... [ paragraph-name-1. [ sentence ] ... ] ... }...
END DECLARATIVES. ]
  [ { section-name-1 SECTION.
    [ sentence ] ... [ paragraph-name-1. ] [ sentence ] ... ]  ... } ... ]
```

Format 2 (without-sections):

```
procedure-division-header
[ sentence ] ...  [ { paragraph-name-1.  [ sentence ]  ...  }  ... ]
```

Format 3 (object-oriented):

```
PROCEDURE DIVISION.
[ { method-definition }  ...  ]
```

where procedure-division-header is:

```
PROCEDURE DIVISION [ using-phrase ] [ RETURNING data-name-2 ]
    [ RAISING { exception-name-1
              { [ FACTORY OF ] class-name-1 } ...  .
              { interface-name-1
    ]
```

where using-phrase is:

```
USING { [ BY REFERENCE ] { [ OPTIONAL ] data-name-1 } ... } ...
      { BY VALUE { data-name-1 } ...                      }
```

where the following meta-language terms are described in the indicated subclauses:

Term	Subclause
method-definition	10.6, COBOL compilation group
sentence	14.5, Procedural statements and sentences
use-statement	14.9.45, USE statement

14.2.2 Syntax rules

FORMATS 1 AND 2

1) Data-name-1 shall be defined as a level 01 entry or a level 77 entry in the linkage section. A particular user-defined word shall not appear more than once as data-name-1. The data description entry for data-name-1 shall not contain a BASED clause or a REDEFINES clause.

   NOTE 1 This restriction for based items does not prohibit passing based items as arguments.

   NOTE 2 A data item defined subsequently in the linkage section may specify REDEFINES data-name-1.

2) Each data-name-1 specified in a BY VALUE phrase shall be defined as a data item of class numeric, object, or pointer.

3) The RETURNING phrase shall be specified in a function definition and in a function prototype definition.

4) The RETURNING phrase may be specified in a method definition, a program definition, or a program prototype.

5) Data-name-2 shall be defined as a level 01 entry or level 77 entry in the linkage section. The data description entry for data-name-2 shall not contain a BASED clause or a REDEFINES clause.

   NOTE 1 This restriction for based items does not prohibit specifying a based item as the returning item in the activating element.

   NOTE 2 A data item defined subsequently in the linkage section may specify REDEFINES data-name-2.

6) Data-name-2 shall not be the same as data-name-1.

7) Exception-name-1 shall be a level-3 exception-name for EC-USER.

8) Class-name-1 shall be the name of a class specified in the REPOSITORY paragraph.

9) Interface-name-1 shall be the name of an interface specified in the REPOSITORY paragraph.

10) Formats 1 and 2 may be specified in a source element if and only if that source element is a function definition, a function prototype definition, a method definition, a program definition, or a program prototype definition.

11) A procedure division in a method prototype shall contain only a procedure division header.

FORMAT 3

12) Format 3 may be specified in a source element if and only if that source element is a factory definition, an instance definition, or an interface definition, but not in a method definition.

13) A procedure division in an interface definition, an instance definition, or a factory definition that is not in a method definition shall be an object-oriented format procedure division.

14.2.3 General rules

FORMATS 1 AND 2

1) Execution begins with the first statement of the procedure division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.

2) The USING phrase identifies the formal parameters used by the function, method, or program for any arguments passed to it. The arguments passed from the activating element are:

   — the arguments specified in the USING phrase of a CALL statement
   — the arguments specified in the USING phrase of an INVOKE statement
   — the arguments specified in an inline invocation of a method
   — the arguments specified in a function reference
   — the argument defined by the rules of an object-property for invocation of an implicit SET property method.

   The correspondence between the arguments and the formal parameters is established on a positional basis.

   The conformance requirements for formal parameters and returning items are specified in 14.8, Conformance for parameters and returning items.

3) If the OPTIONAL phrase is specified for data-name-1, the OMITTED phrase may be specified as the corresponding argument; otherwise, the OMITTED phrase shall not be specified as the corresponding argument.

4) Both the BY REFERENCE and the BY VALUE phrases are transitive across the parameters that follow them until another BY REFERENCE or BY VALUE phrase is encountered. If neither the BY REFERENCE nor the BY VALUE phrase is specified prior to the first parameter, the BY REFERENCE phrase is assumed.

5) Data-name-1 is a formal parameter for the function, method, or program.

6) Data-name-2 is the name used in the function, method, or program for the result that is returned to the activating element according to 14.6.5, Results of runtime element execution.

   NOTE In COBOL, the storage for the returning item is allocated in the activating source unit. The activated element contains only a formal description in its linkage section.

7) The initial value of the returning item, data-name-2, is undefined.

8) If the argument is passed by reference, the activated runtime element operates as if the formal parameter occupies the same storage area as the argument.

9) If the argument is passed by content, the activated runtime element operates as if the record in the linkage section were allocated by the activating runtime element during the process of initiating the activation and as if this record does not occupy the same storage area as the argument in the activating runtime element.

   If the activated runtime element is a program for which there is no program-specifier in the REPOSITORY paragraph of the activating runtime element and there is no NESTED phrase specified on the CALL statement, this allocated record is of the same length as the argument, where the maximum length is used if the argument is described as a variable-occurrence data item. That argument is moved to this allocated record without conversion. This record is then treated by the activated runtime element as if it were the argument and as if it were passed by reference.

   If the activated runtime element is one of the following:

— a program for which there is a program-specifier in the REPOSITORY paragraph of the activating runtime element
— a program and the NESTED phrase is specified on the CALL statement
— a method
— a function

then this allocated record is

— a data item of the same category, usage, and length as the argument, if the formal parameter is described with the ANY LENGTH clause,
— otherwise, a data item with the same description and the same number of bytes as the formal parameter, where the maximum length is used if the formal parameter is described as a variable-occurrence data item.

The argument is used as the sending operand and the allocated record as the receiving operand in the following:

— if the formal parameter is numeric, a COMPUTE statement without the ROUNDED phrase
— if the formal parameter is of class index, object, or pointer, a SET statement
— otherwise, a MOVE statement.

The allocated record is then treated as if it were the argument and it were passed by reference.

10) If the argument is passed by value, the activated runtime element operates as if the record in the linkage section were allocated by the activating runtime element during the process of initiating the activation.

The allocated record is a data item of the same description as the formal parameter. The argument is used as the sending operand and the allocated record, whose description is specified in the linkage section, is used as the receiving operand in the following:

— if the formal parameter is numeric, a COMPUTE statement without the ROUNDED phrase
— if the formal parameter is of class object or pointer, a SET statement.

The activated runtime element is given access to the allocated record.

11) At all times in the activated element, references to data-name-1 and to data-name-2 are resolved in accordance with their description in the linkage section.

12) Exception-name-1 specifies an exception that this runtime element may raise in an EXIT or GOBACK statement, in addition to the exception-names defined in table 14, Exception-names and exception conditions. If class-name-1 is specified, an object of class-name-1 or from a subclass of class-name-1 may be raised by EXIT or GOBACK statements within this runtime element. If interface-name-1 is specified, an object that implements interface-1 may be raised by EXIT or GOBACK statements within this runtime element.

13) When either the activating or the activated runtime element is other than a COBOL runtime element, the implementor shall specify the restrictions and mechanisms for all supported language products.

NOTE The details of these restrictions and mechanisms might include parameter matching, representation of a data type, return of a value, and omission of parameters.

14.3 Declaratives

A Declarative is a procedure that is to be executed when a specific exception or condition occurs based on the USE statement. Declarative sections shall be grouped at the beginning of the procedure division preceded by the keyword DECLARATIVES and followed by the keywords END DECLARATIVES.

The sections specified between the keywords DECLARATIVES and END DECLARATIVES constitute the declarative portion of a source element. All other sections in a source element constitute the nondeclarative portion.

14.4 Procedures

A procedure is composed of a paragraph, or a group of successive paragraphs, or a section, or a group of successive sections within the procedure division. If one paragraph is in a section, all paragraphs shall be in sections. A procedure-name is a word used to refer to a paragraph or section in the source element in which it occurs. It consists of a paragraph-name (that may be qualified) or a section-name.

14.4.1 Sections

A section consists of a section header followed by zero, one, or more successive paragraphs. A section ends immediately before the next section or at the end of the procedure division or, in the declaratives portion of the procedure division, at the keywords END DECLARATIVES.

14.4.2 Paragraphs

A paragraph consists of a paragraph-name followed by a separator period and by zero, one, or more successive sentences or, if the paragraph-name is omitted, one or more successive sentences following the procedure division header or a section header. A paragraph ends immediately before the next paragraph-name or section-name or at the end of the procedure division or, in the declaratives portion of the procedure division, at the keywords END DECLARATIVES.

14.5 Procedural statements and sentences

A procedural statement is a unit of the COBOL language that specifies an action to be taken. Statement names are identified in table 13, Procedural statements.

Within the procedure division, there are the following types of statements:

— declarative statements, which specify actions that may be taken during the processing of other statements
— imperative statements, which specify unconditional actions
— conditional statements, which specify, or contain one or more phrases that specify, actions that depend on the truth value of a condition.

A declarative statement begins with the statement name USE and directs that actions be taken in response to specified conditions encountered during the processing of other statements.

An imperative statement specifies an unconditional action to be taken by the runtime element or is a conditional statement that is delimited by its explicit scope terminator, as specified in table 13, Procedural statements.

A conditional statement specifies that the truth value of a condition is evaluated and used to determine subsequent flow of control. Any statement with a conditional phrase that is not terminated by its explicit scope terminator is a conditional statement.

A sentence is a sequence of one or more procedural statements, the last of which is terminated by a separator period.

Wherever 'imperative-statement' appears in the general format of a statement, 'imperative-statement' refers to one or more imperative statements ended either by a separator period or by any phrase associated with that general format.

Table 13 — Procedural statements

Statement name	Conditional phrase	Explicit scope terminator
ACCEPT	[NOT] ON EXCEPTION	END-ACCEPT
ADD	[NOT] ON SIZE ERROR	END-ADD

Table 13 — Procedural statements (Continued)

Statement name	Conditional phrase	Explicit scope terminator
ALLOCATE		
CALL	ON OVERFLOW [NOT] ON EXCEPTION	END-CALL
CANCEL		
CLOSE		
COMPUTE	[NOT] ON SIZE ERROR	END-COMPUTE
CONTINUE		
DELETE	[NOT] INVALID KEY	END-DELETE
DISPLAY	[NOT] ON EXCEPTION	END-DISPLAY
DIVIDE	[NOT] ON SIZE ERROR	END-DIVIDE
EVALUATE	WHEN	END-EVALUATE
EXIT		
FREE		
GENERATE		
GOBACK		
GO TO		
IF	THEN ELSE	END-IF
INITIALIZE		
INITIATE		
INSPECT		
INVOKE		
MERGE		
MOVE		
MULTIPLY	[NOT] ON SIZE ERROR	END-MULTIPLY
OPEN		
PERFORM		END-PERFORM
RAISE		
READ	[NOT] AT END [NOT] INVALID KEY	END-READ
RELEASE		
RESUME		
RETURN	[NOT] AT END	END-RETURN
REWRITE	[NOT] INVALID KEY	END-REWRITE
SEARCH	WHEN AT END	END-SEARCH

Table 13 — Procedural statements (Continued)

Statement name	Conditional phrase	Explicit scope terminator
SET		
SORT		
START	[NOT] INVALID KEY	END-START
STOP		
STRING	[NOT] ON OVERFLOW	END-STRING
SUBTRACT	[NOT] ON SIZE ERROR	END-SUBTRACT
SUPPRESS		
TERMINATE		
UNLOCK		
UNSTRING	[NOT] ON OVERFLOW	END-UNSTRING
VALIDATE		
WRITE	[NOT] INVALID KEY [NOT] END-OF-PAGE	END-WRITE

14.5.1 Conditional phrase

A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from the execution of a conditional statement.

The conditional phrases are specified in table 13, Procedural statements.

14.5.2 Scope of statements

A procedure division statement's scope is the range of words and separators that constitute the statement. The scope of a statement begins with the first word of the statement name and continues until the statement is either explicitly or implicitly terminated, and includes any procedural statements occurring syntactically between the start and termination of that statement.

14.5.2.1 Explicit scope termination

The scope of a statement may be explicitly terminated by using its associated scope terminator as specified in table 13, Procedural statements. A statement written with its explicit scope terminator is a subset of imperative statements and is termed a delimited scope statement.

An explicit scope terminator terminates the scope of:

1) the most-recently preceding unterminated statement having the statement-name for which that scope terminator is defined, and

2) any unterminated statements that appear between that statement-name and the explicit scope terminator.

14.5.2.2 Implicit scope termination

The scope of a statement that is not explicitly terminated is implicitly terminated as follows:

1) for a single imperative statement not contained within another statement, by

    a) any element that follows the exhaustion of the statement's syntax, or

b) the next-encountered statement-name, or

c) a separator period.

2) for a single imperative statement contained within another statement, by

a) any element that terminates an imperative statement not contained within another statement,

b) the termination of the scope of any containing statement, or

c) the next phrase of any containing statement.

3) for a conditional statement not contained within another statement, by a separator period.

4) for a conditional statement contained within another statement, by

a) the termination of any containing statement, or

b) the next phrase of any containing statement.

Any phrase encountered is the next phrase of the most-recently preceding unterminated statement with which that phrase may be syntactically associated. If all permitted occurrences of a phrase have already been specified for a given statement, a subsequent occurrence of that phrase is not syntactically associated with that statement. An unterminated statement is any statement that has begun but has not yet been either explicitly or implicitly terminated.

A contained statement is also referred to as a nested statement.

14.6 Execution

14.6.1 Run unit organization

At runtime, the highest-level unit of a COBOL application is the run unit. A run unit is an independent entity that may be executed without communicating with, or being coordinated with, any other run unit except that it may process files or set and test switches that were written or will be read by other run units. A run unit contains one or more runtime modules.

A runtime module results from compiling a compilation unit. Each runtime module contains one or more runtime elements.

A runtime element results from the compilation of a function, method, or program. When a runtime element is activated, parameters upon which it is to operate may be passed to it by the runtime element that calls it.

A run unit and each of its contained runtime modules may also contain resources and data storage areas needed for the execution and intercommunication of the runtime elements contained in the run unit.

A run unit may additionally contain runtime modules and data storage areas derived from the compilation of compilation units written in languages other than COBOL; in this case the requirements for the relationship and interaction between the COBOL and the non-COBOL modules are defined by the implementor.

14.6.2 State of a function, method, object, or program

14.6.2.1 State of a function, method, or program

The state of a function, method, or program at any point in time in a run unit may be active or inactive. When a function, method, or program is activated, its state may also be initial or last-used.

14.6.2.1.1 Active state

A function, method, or program may be activated recursively. Therefore, several instances of a function, method, or program may be active at once. When a rule indicates that the active state of a runtime element is tested, it is in the active state if any instance of the element is active.

An instance of a function is placed in an active state when it is successfully activated and remains active until the execution of a GOBACK or STOP statement or an implicit or explicit function format of the EXIT statement within this instance of this function.

An instance of a method is placed in an active state when it is successfully activated and remains active until the execution of a GOBACK or STOP statement or implicit or explicit method format of the EXIT statement within this instance of this method.

An instance of a program is placed in an active state when it is successfully activated by the operating system or successfully called from a runtime element. An instance of a program remains active until the execution of one of the following:

— a STOP statement

— in a called program, an implicit or explicit program format of an EXIT statement within that program

— a GOBACK statement within either that same called program or a program that is not under the control of a calling runtime element.

Whenever an instance of a function, method, or program is activated, the control mechanisms for all PERFORM statements contained in that instance of the function, method, or program are set to their initial states and the initial program collating sequences are in effect.

14.6.2.1.2 Initial and last-used states of data

When a function, method, or program is activated, the data within is in either the initial state or the last-used state.

14.6.2.1.2.1 Initial state

Automatic data and initial data is placed in the initial state every time the function, method, or program in which it is described is activated.

Static data is placed in the initial state:

1) The first time the function, method, or program in which it is described is activated in a run unit.

2) The first time the program in which it is described is activated after the execution of an activating statement referencing a program that possesses the initial attribute and directly or indirectly contains the program.

3) The first time the program in which it is described is activated after the execution of a CANCEL statement referencing the program or CANCEL statement referencing a program that directly or indirectly contains the program.

When data in a function, method, or program is placed in the initial state, the following occurs:

1) The internal data described in the working-storage section and local-storage section is initialized as described in 13.18.62, VALUE clause.

2) The function, method, or program's internal file connectors are initialized by setting them to not be in any open mode.

3) The attributes of screen items are set as specified in the screen description entry.

4) The address of each based item is set to null.

5) For each dynamic-capacity table, except where the table is defined by an elementary entry with a VALUE clause, the capacity of the table is set to the minimum capacity specified in the corresponding OCCURS clause. If the INITIALIZED keyword is present in the OCCURS clause, all the occurrences, if any, of the table are then initialized.

6) The length of each any-length elementary item that is specified without a VALUE clause is set to zero.

Before data is placed in the initial state, the initial alphanumeric and national program collating sequences are determined as specified in 12.3.5, OBJECT-COMPUTER paragraph.

14.6.2.1.2.2 Last-used state

Static and external data are the only data that are in the last-used state. External data is always in the last-used state except when the run unit is activated. Static data is in the last-used state except when it is in the initial state as defined above.

14.6.2.2 Initial state of object data

The initial state of an object is the state of the object immediately after it is created. Internal data, internal file connectors, and the attributes of screen items are initialized in the same manner as when data in a function, method, or program is placed in the initial state, in accordance with 14.6.2.1.2.1, Initial state.

Before object data is placed in the initial state, the initial alphanumeric and national program collating sequences are determined as specified in 12.3.5, OBJECT-COMPUTER paragraph.

14.6.3 Explicit and implicit transfers of control

The flow of control mechanism transfers control from statement to statement in the sequence in which they were written unless an explicit transfer of control overrides this sequence or there is no next executable statement to which control may be passed. The transfer of control from statement to statement occurs without the writing of an explicit procedure division statement, and, therefore, is an implicit transfer of control.

COBOL provides both explicit and implicit means of altering the implicit control transfer mechanism.

In addition to the implicit transfer of control between consecutive statements, implicit transfer of control also occurs when the normal flow is altered without the execution of a procedure branching statement. COBOL provides the following types of implicit control flow alterations that override the statement-to-statement transfers of control:

1) If a paragraph is being executed under control of another COBOL statement such as PERFORM, USE, SORT, and MERGE, and the paragraph is the last paragraph in the range of the controlling statement, then an implied transfer of control occurs from the last statement in the paragraph to the control mechanism of the last executed controlling statement. Further, if a paragraph is being executed under the control of a PERFORM statement that causes iterative execution, and that paragraph is the first paragraph in the range of that PERFORM statement, an implicit transfer of control occurs between the control mechanism associated with that PERFORM statement and the first statement in that paragraph for each iterative execution of the paragraph.

2) When a SORT or MERGE statement is executed, an implicit transfer of control occurs to any associated input or output procedures.

3) When any COBOL statement is executed that results in the execution of a declarative section, an implicit transfer of control to the declarative section occurs.

   NOTE Another implicit transfer of control occurs after execution of the declarative section, as described in rule 1 above.

An explicit transfer of control consists of an alteration of the implicit control transfer mechanism by the execution of a procedure branching or conditional statement. An explicit transfer of control may be caused only by the execution of a procedure branching or conditional statement. The procedure branching statement EXIT PROGRAM causes an explicit transfer of control only when the statement is executed in a called program.

If control is transferred either implicitly or explicitly to a paragraph containing no statements, execution proceeds as if the paragraph contained only a single sentence consisting of a CONTINUE statement.

In this document, the term 'next executable statement' is used to refer to the next COBOL statement to which control is transferred according to the rules above and the rules associated with each language element.

There is no next executable statement when the execution of an EXIT FUNCTION, EXIT METHOD, EXIT PROGRAM, GOBACK, or STOP statement transfers control outside the COBOL source element.

In the declarative section, there is no next executable statement after either the last statement when the paragraph in which it appears is not being executed under the control of some other COBOL statement, or the last statement when the statement is in the range of an active PERFORM statement executed in a different section and this last statement of the declarative section is not also the last statement of the procedure that is the exit of the active PERFORM statement. In these cases the flow of control is undefined.

There is also no next executable statement after the last statement in a source element when the paragraph in which it appears is not being executed under the control of some other COBOL statement in that source element, after the end marker, and if there are no procedure division statements in a program, function, or method. In these cases, an implicit GOBACK statement without any optional phrases is executed.

14.6.4 Item identification

Item identification is the process of identifying a specific data item referenced by an identifier by evaluating all of that identifier's references. If a step in the evaluation of an identifier requires evaluation of another identifier or an arithmetic expression, that evaluation is done in full before proceeding to the next step. The item identification steps that are applicable to that identifier are evaluated in the following order:

1) locale identification

2) function evaluation

3) inline method invocation

4) subscript evaluation

5) object property evaluation

6) length evaluation for an occurs-depending group item

7) reference modification

Unless otherwise specified, item identification is done for an identifier as the first step in the evaluation of that identifier and the identifiers within a statement are evaluated in left to right order as the first operation of the execution of that statement.

14.6.5 Results of runtime element execution

The result of the execution of a program, function, or method that specifies a RETURNING phrase in its procedure division header, is the content of the data item referenced by that RETURNING phrase.

The result becomes available to the activating element after the activated element returns as follows:

— If the runtime element is activated by a CALL or INVOKE statement, the result is placed in the data item referenced by that RETURNING phrase of that activating statement.

— If the runtime element is activated by a function-identifier or an inline method invocation, the result is placed in the temporary data item referenced by that identifier.

14.6.6 Locale identification

Locale identification is the process of identifying a specific locale for use in locale-based processing. Locale identification occurs as follows:

1) At run unit activation, the user default locale becomes the current locale for the run unit for all locale categories.

   NOTE The implementor specifies whether a locale set by COBOL is recognized in a non-COBOL runtime module, and whether a locale set by a non-COBOL runtime module is recognized in COBOL. In order to use a locale set by a non-COBOL runtime module, it is necessary to execute a SET statement specifying the USER-DEFAULT phrase.

2) On activation of a runtime element, if the CHARACTER CLASSIFICATION clause is specified in the OBJECT-COMPUTER paragraph, category LC_CTYPE in the specified locale is used for character classification in the class test and the UPPER-CASE and LOWER-CASE intrinsic functions.

3) When execution of a set-locale format SET statement switches the locale for one or more locale categories, the new locale becomes the current locale for the run unit for the switched categories; the current locale remains unchanged for categories that are not switched.

4) If an alphabet-name specified in the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph is associated with a locale, category LC_COLLATE in the associated locale is used for comparisons and ordering as specified in 12.3.5, OBJECT-COMPUTER paragraph.

5) For a SORT or MERGE statement specifying an alphabet-name associated with a locale in the COLLATING SEQUENCE phrase, category LC_COLLATE in the associated locale is used for that statement. A locale switch during execution of a SORT or MERGE statement has no effect on the processing of that SORT or MERGE statement.

6) For a data item described with a LOCALE phrase, if a locale-name is specified, locale category LC_MONETARY in that locale is used for editing and de-editing of the data item. If a locale-name is not specified, category LC_MONETARY in the locale current at the time of editing or de-editing is used.

7) For a LOCALE-COMPARE intrinsic function specifying a locale as an argument, category LC_COLLATE from the specified locale is used for evaluation of that function-identifier; if a locale is not specified as an argument, category LC_COLLATE from the current locale is used.

8) For a LOCALE-DATE or LOCALE-TIME intrinsic function specifying a locale as an argument, category LC_TIME from the specified locale is used for evaluation of that function-identifier; if a locale is not specified as an argument, category LC_TIME from the current locale is used.

9) Upon return of control from another COBOL runtime element, the locale in effect for each locale category at the time of exit from the returning runtime element becomes the current locale for that category.

   NOTE The intended behavior might be that the invoked program, function, or method switches locale for its caller; if not, the invoked program, function, or method is responsible for saving the locale on entry and restoring it before returning.

14.6.7 Sending and receiving operands

An operand is a sending operand if its contents prior to the execution of a statement may be used by the execution of the statement. An operand is a receiving operand if its contents may be changed by the execution of the statement. Operands may be referenced either explicitly or implicitly by a statement. For some statements, an

operand is both a sending operand and a receiving operand. The rules for a statement specify whether operands are sending operands, receiving operands, or both when this is not clear from the context of the statement.

14.6.8 Alignment of data within data items

The standard rules for positioning data within an elementary item depend on the category of the receiving item. These rules are:

1) If the receiving data item is described as a fixed-point numeric item:

   a) If the sending data item is described with an IEEE floating-point usage, the value is treated as if it had been converted to a fixed-point value.

   b) The data is aligned by decimal point and is moved to the receiving digit positions with zero fill or truncation on either end as required.

   c) When an assumed decimal point is not explicitly specified, the data item is treated as if it has an assumed decimal point immediately following its rightmost digit and is aligned as in rule 1b.

2) If the receiving data item is described as a floating-point numeric item, the alignment of the data is specified by the implementor.

3) If the receiving data item is a fixed-point numeric-edited data item and the LOCALE phrase of the PICTURE clause is not specified in its data description entry, the data moved to the edited data item is aligned by decimal point with zero fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.

   If the receiving data item is a numeric-edited data item and the LOCALE phrase of the PICTURE clause is specified in its data description entry, alignment and zero fill or truncation take place as described in 13.18.39.4, Editing rules.

4) If the receiving data item is a floating-point numeric-edited data item and the value to be edited is not zero, the data moved to the edited data item is aligned so that the leftmost digit is not zero.

5) If the receiving data item is alphabetic, alphanumeric, alphanumeric-edited, national, or national-edited, the sending data shall be moved, after any specified conversion, to the receiving character positions and aligned at the leftmost character position in the data item with space fill or truncation to the right, as required. If the JUSTIFIED clause is specified for the receiving item, alignment differs as specified in 13.18.31, JUSTIFIED clause.

   NOTE If the sending data item or literal is zero-length, the entire receiving data item is space filled.

6) If the receiving data item is boolean, the sending data is placed, after any conversion, in the receiving boolean positions and aligned at the leftmost boolean position in the data item with zero fill or truncation to the right, as required. If the JUSTIFIED clause is specified for the receiving item, alignment differs as specified in 13.18.31, JUSTIFIED clause.

   NOTE 1 When an item is of usage bit, the item is not necessarily aligned on a byte boundary and the item need not occupy an integral number of bytes.

   NOTE 2 If the sending data item or literal is zero-length, the entire receiving data item is zero filled.

14.6.9 Overlapping operands

When the data items referenced by a sending and a receiving operand in any statement are identified as sharing either a part of or all of their storage areas, and the rules for the statement do not provide for a specific result in the following circumstances, then:

1) When the data items are not described by the same data description entry, the result of the statement is undefined.

2) When the data items are described by the same data description entry, the result of the statement is the same as if the data items shared no part of their respective storage areas.

NOTE The execution of a statement may have side effects on implicitly referenced items. When such items share a part or all of their storage areas with other explicitly or implicitly referenced items, there may be unexpected results.

Similarly, when the rules for a statement specify multiple receiving operands, and some or all of those receiving operands share a part or all of their storage areas, there may be unexpected results among those receiving operands.

This topic is further discussed at D.5, Sharing of storage among data items.

In the case of reference modification, the unique data item produced by reference modification is not considered to be the same data description entry as any other data description entry. Therefore, if an overlapping situation exists, the results of the operation are undefined.

14.6.10 Normal run unit termination

When normal run unit termination occurs, the runtime system performs the following:

1) An implicit CLOSE statement without any phrases is executed for each file that is in the open mode. These implicit close statements shall be executed for all open files in the run unit, even when an error occurs during the execution of one or more of the CLOSE statements. Any declaratives associated with these files are not executed.

2) Any storage obtained with an ALLOCATE statement and not yet released by a FREE statement is released.

3) All instance objects are destroyed.

NOTE Any open files in an object are closed before the object is deleted.

4) If a locale was in use in the run unit, the implementor specifies whether the locale is reset to the locale in effect when the run unit was activated.

5) Any resources occupied by dynamic-capacity tables or any-length elementary items are freed.

14.6.11 Abnormal run unit termination

When abnormal run unit termination occurs, the runtime system attempts to perform the operations of normal termination as specified in 14.6.10, Normal run unit termination. The circumstances of abnormal termination may be such that execution of some or all of these operations is not possible. The runtime system performs all operations that are possible.

The operating system shall indicate an abnormal termination of the run unit if such a capability exists within the operating system.

14.6.12 Exception condition handling

14.6.12.1 Exception conditions

An exception condition is either a condition associated with a specific exception status indicator or an exception object. An exception object is any object that is raised by the execution of either a RAISE statement or an EXIT or a GOBACK statement in which the object is specified in the RAISING phrase. An exception status indicator is a conceptual entity that exists for each function, method, or program for each exception condition and has two states, set or cleared. The initial state of all exception status indicators is cleared. An exception status indicator is set when the associated exception exists. Associated with each exception status indicator are one or more

exception-names. These exception names, together with the interface-names and class-names of exception objects are used to enable checking for the exception condition, to specify the action to be taken when the exception is raised, and to determine which exception condition caused an exception declarative to be executed.

In addition to exception status indicators, a last exception status exists for the entire run unit. It is a conceptual entity that is set to indicate the last level-3 exception condition that was raised in the run unit, the fact that an exception object was raised, or that no exception condition exists. The SET statement can be used to set the indicator to no exception condition exists. The last exception status can be interrogated with the EXCEPTION-STATUS function. Associated with this last exception status is information that is accessed by the EXCEPTION-FILE, EXCEPTION-FILE-N, EXCEPTION-LOCATION, EXCEPTION-LOCATION-N, and EXCEPTION-STATEMENT functions.

Exception conditions differ from exception processing that exists for specific facilities or statements. Input-output statements have an I-O status associated with file connectors. I-O status is not an exception condition. When checking for EC-I-O exception conditions is enabled, the associated exception conditions are raised based on the resulting I-O status value, but if checking for EC-I-O exception conditions is not enabled, there is no link between EC-I-O exception conditions and I-O status values.

If checking for an exception condition is enabled and an exception status indicator is set as a result of an exception detected during the execution of a statement, the associated exception condition is raised, the last exception status is set to indicate that exception condition, and the predefined object reference EXCEPTION-OBJECT is set to null. Unless otherwise specified, if more than one exception is detected during the execution of a statement, the one that is set to exist is undefined. The execution of the statement may be successful or unsuccessful, depending on the rules for the statement, the fatality of the exception condition as specified in the table of exception-names, and implementor-defined actions. If no exception is detected during the execution of a statement or if checking for an exception that occurs is not enabled, no exception condition is raised. If an exception condition is raised and an associated exception declarative is executed, a reference to the EXCEPTION-STATUS function within that declarative returns identifying information associated with the exception condition that caused the declarative to be executed. All exception status indicators are cleared at the beginning of the execution of any statement.

NOTE 1 While I-O status indicators correspond to exception conditions, I-O status indicators are independent from this condition handling facility. I-O status indicators are always enabled and cannot be turned off.

NOTE 2 The fatality of an exception condition is either fatal or nonfatal. Exception conditions that would cause corrupted data, undefined execution paths, and incorrect results are specified as fatal. Other exception conditions are specified as nonfatal.

Exception-names are organized into a hierarchy of three levels for purposes of enabling checking, selecting a declarative, and reporting the exception that occurred. The highest level, level-1, is the exception-name EC-ALL. Level-2 consists of the exception-names EC-ARGUMENT, EC-BOUND, EC-DATA, EC-FLOW, EC-I-O, EC-IMP, EC-LOCALE, EC-OO, EC-ORDER, EC-OVERFLOW, EC-PROGRAM, EC-RANGE, EC-RAISING, EC-REPORT, EC-SCREEN, EC-SIZE, EC-SORT-MERGE, EC-STORAGE, EC-USER, and EC-VALIDATE. The lowest level, level-3, consists of the level-2 names suffixed by a hyphen and additional characters. Only the lowest level exception-names are associated with exception status indicators. The implementor is not required to raise any exception conditions for level-3 exception-names that are associated with an optional language elements or processor-dependent language elements that the implementor has not implemented unless the description of that language element in A.3, Processor-dependent language element list or A.4, Optional language element list, requires that an exception condition be raised.

The level-3 exception-names for implementor-defined exceptions are defined by the implementor by creating an exception-name starting with the characters 'EC-IMP-' and ending with a suffix containing only basic letters, basic digits, and the hyphen and underscore basic special characters. The implementor defines the action to be taken, the fatality, and when the exception is raised.

The level-3 exceptions-names for user-defined exceptions shall start with the characters 'EC-USER-' and end with a suffix containing only characters selected from the set of basic letters, basic digits, and the hyphen and underscore basic special characters. The hyphen or underscore shall not appear as the last character of the suffix. The name is defined by specifying it anywhere that an exception-name may be specified. All user-defined exception conditions shall be nonfatal. Checking is not enabled unless the TURN directive is used to enable

checking for the exception. A user-defined exception condition may be raised only by a RAISE statement or an EXIT or GOBACK statement with the RAISING phrase.

All exception-names may be specified in TURN compiler directives to enable or disable checking for a specific exception condition or exception conditions subordinate to the exception condition that is specified. By default, checking is not enabled for any exception condition. During the execution of any statement, if checking for an exception condition is not enabled, the exception condition will not be raised, even if the events that normally raise the exception condition occur. Therefore, when the general rules for a statement indicate that a specific exception condition exists, it is raised only if checking for that exception condition is enabled.

An exception object is a nonfatal exception condition; other exception conditions are either fatal or nonfatal as indicated in table 14, Exception-names and exception conditions.

14.6.12.1.1 Normal completion of a declarative procedure

A declarative procedure is said to complete normally if, during execution of the declarative procedure, none of the following occur:

1) An EXIT FUNCTION, EXIT METHOD, EXIT PROGRAM, GOBACK, RESUME, or STOP statement that is specified in this function, method, or program is executed within the scope of the declarative.

2) Any directly or indirectly activated runtime element terminates the run unit.

14.6.12.1.2 Fatal exception conditions

If a fatal exception condition exists, processing of the statement is interrupted and one of the following occurs in the order specified:

1) If a conditional phrase without the NOT phrase is specified in the interrupted statement and the rules for that statement indicate that the fatal exception condition is to be processed by the conditional phrase, the procedures for nonfatal exceptions as specified in 14.6.12.1.3, Non fatal exception conditions, apply.

2) If checking for the exception condition is enabled and there is an applicable USE statement in the source unit that specifies the exception-name associated with the exception condition or an exception-name of a higher level in the same hierarchy, the associated declarative is executed. If execution of the declarative completes normally the execution of the run unit is terminated abnormally as specified in 14.6.11, Abnormal run unit termination.

3) If checking for the exception condition is enabled, and the exception condition is neither EC-FLOW-GLOBAL-EXIT nor EC-FLOW-GLOBAL-GOBACK, and there is an applicable PROPAGATE ON directive, the exception condition is propagated as if a GOBACK statement with the RAISING LAST EXCEPTION phrase were executed.

4) If checking for the exception condition is enabled, execution of the run unit is terminated abnormally as specified in 14.6.11, Abnormal run unit termination.

5) If checking for the exception condition is not enabled, the implementor defines whether or not execution will continue, how it will continue, and how any receiving operands are affected.

If checking is not enabled for a fatal exception condition and the exception condition is detected by the compiler, the implementor is not required to produce executable code. It is implementor-defined which fatal exception conditions, if any, are detected at compile time, and the circumstances under which they are detected.

14.6.12.1.3 Non fatal exception conditions

If a nonfatal exception condition other than an exception object is set to exist, processing depends on whether or not checking for that exception condition is enabled. If it is not enabled, processing proceeds as indicated for the statement in which the exception condition was set to exist. If there are no specific rules, the exception condition

is ignored and processing is as if the exception condition was not set to exist. If checking for that exception condition is enabled, processing of the statement is interrupted and one of the following occurs in the order specified:

1) If a conditional phrase without the NOT phrase is specified in the interrupted statement, the imperative statement associated with that conditional phrase is executed as specified in the rules for the specific statement.

2) If there is an applicable USE statement in the source unit that specifies the exception-name associated with the exception condition or an exception-name of a higher level in the same hierarchy, the associated declarative is executed. If execution of the declarative completes normally, execution continues as specified in the statement for normal execution. If a conditional phrase with a NOT phrase is specified in the interrupted statement, imperative-statement in that phrase is not executed.

3) Execution of the statement continues as specified in the rules for that statement.

14.6.12.1.4 Exception objects

When an exception object is raised, the following occurs:

1) The predefined object reference EXCEPTION-OBJECT is set to the content of the object reference specified in the RAISE statement or the RAISING phrase of the EXIT or GOBACK statement that caused the exception object to be raised.

2) The last exception status is set to indicate that an exception object has been raised.

If an exception object is raised by a RAISE statement, the associated declarative is executed. If execution of the declarative completes normally, execution continues with the statement following the RAISE statement. If there is no associated declarative, execution continues as specified in the RAISE statement.

If an exception object is raised by an EXIT or GOBACK statement, one of the following occurs:

1) If the exception object is neither of the following:

   a) an object whose class is specified or whose class is a subclass of a class specified in the RAISING phrase of the procedure division header of the source element containing this EXIT or GOBACK statement and the presence or absence of the FACTORY phrase is the same in the description of the object reference raised by the EXIT or GOBACK statement as in the RAISING phrase of the procedure division header of the source element that contains this EXIT statement, or

   b) an object that implements an interface specified in the RAISING phrase of the procedure division header of the source element containing this EXIT or GOBACK statement;

   execution of the EXIT or GOBACK statement is as if EXCEPTION EC-OO-EXCEPTION were specified in the RAISING phrase of the EXIT or GOBACK statement instead of an exception object and processing continues as specified in 14.6.12.1.2, Fatal exception conditions.

2) Otherwise, if a USE statement in the activating runtime element specifies an applicable class or interface, the associated declarative is executed. If execution of the declarative completes normally, execution continues as specified in the activating statement for normal execution.

3) Otherwise, if a PROPAGATE ON directive is in effect for the activating runtime element, the exception is propagated as if a GOBACK statement with the RAISING LAST EXCEPTION phrase were specified in this activating runtime element. However, if no applicable class or interface is specified in the RAISING phrase of the procedure division header in the activating element, the RAISING phrase is EXCEPTION EC-OO-EXCEPTION, instead of LAST EXCEPTION.

4) Otherwise, execution of the EXIT or GOBACK statement in the activated element is as if EXCEPTION EC-OO-EXCEPTION were specified in the RAISING phrase, instead of an exception object.

14.6.12.1.5 Exception-names and exception conditions

Table 14, Exception-names and exception conditions, is a list of the exception-names and their attributes. The meaning of the columns in the table are:

Exception-name - The exception-name associated with an exception condition or a hierarchy of exception-names.

Cat - The category of the exception condition: nonfatal (NF), fatal (Fatal), or implementor-defined (Imp). The category of a level-1 or level-2 exception condition is that of the level-3 exception condition that was raised.

Description - A brief description of what the exception condition means.

Table 14 — Exception-names and exception conditions

Exception-name	Cat	Description
EC-ALL		Any exception
EC-ARGUMENT		Argument error
EC-ARGUMENT-FUNCTION	Fatal	Function argument error
EC-ARGUMENT-IMP	Imp	Implementor-defined argument error
EC-BOUND		Boundary violation
EC-BOUND-IMP	Imp	Implementor-defined boundary violation
EC-BOUND-ODO	Fatal	OCCURS ... DEPENDING ON data item out of bounds
EC-BOUND-OVERFLOW	NF	Current capacity of dynamic-capacity table greater than expected value
EC-BOUND-PTR	Fatal	Data-pointer contains an address that is out of bounds
EC-BOUND-REF-MOD	Fatal	Reference modifier out of bounds
EC-BOUND-SET	NF	Invalid use of SET to set capacity of dynamic-capacity table above specified maximum
EC-BOUND-SUBSCRIPT	Fatal	Subscript out of bounds
EC-BOUND-TABLE-LIMIT	Fatal	Capacity of dynamic-capacity table would exceed implementor's maximum
EC-DATA		Data exception
EC-DATA-CONVERSION	NF	Conversion failed because of incomplete character correspondence
EC-DATA-IMP	Imp	Implementor-defined data exception
EC-DATA-INCOMPATIBLE	Fatal	Incompatible data exception
EC-DATA-INTEGRITY	Fatal	Variable-length data item internal format is corrupt
EC-DATA-PTR-NULL	Fatal	Based item data-pointer is set to NULL when referenced
EC-FLOW		Execution control flow violation
EC-FLOW-GLOBAL-EXIT	Fatal	EXIT PROGRAM in a global Declarative
EC-FLOW-GLOBAL-GOBACK	Fatal	GOBACK in a global declarative

Table 14 — Exception-names and exception conditions (Continued)

Exception-name	Cat	Description
EC-FLOW-IMP	Imp	Implementor-defined control flow violation
EC-FLOW-RELEASE	Fatal	RELEASE not in range of SORT
EC-FLOW-REPORT	Fatal	GENERATE, INITIATE, or TERMINATE during USE BEFORE REPORTING declarative
EC-FLOW-RETURN	Fatal	RETURN not in range of MERGE or SORT
EC-FLOW-SEARCH	Fatal	Invalid use of SET to change capacity of dynamic-capacity table during SEARCH of same table
EC-FLOW-USE	Fatal	A USE statement caused another to be executed
EC-FUNCTION		Function exception
EC-FUNCTION-NOT-FOUND	Fatal	Function pointer does not point to a function
EC-FUNCTION-PTR-INVALID	Fatal	Signature mismatch
EC-FUNCTION-PTR-NULL	Fatal	Function pointer used in calling a function is NULL
EC-I-O		Input-output exception
EC-I-O-AT-END	NF	I-O status "1x"
EC-I-O-EOP	NF	An end of page condition occurred
EC-I-O-EOP-OVERFLOW	NF	A page overflow condition occurred
EC-I-O-FILE-SHARING	NF	I-O status "6x"
EC-I-O-IMP	Imp	I-O status "9x"
EC-I-O-INVALID-KEY	NF	I-O status "2x"
EC-I-O-LINAGE	Fatal	The value of a data item referenced in the LINAGE clause is not within the required range
EC-I-O-LOGIC-ERROR	Fatal	I-O status "4x"
EC-I-O-PERMANENT-ERROR	Fatal	I-O status "3x"
EC-I-O-RECORD-OPERATION	NF	I-O status "5x"
EC-IMP		Implementor-defined exception condition
EC-IMP-*suffix* (implementor specifies *suffix*)	Imp	Level-3 implementor-defined exception condition
EC-LOCALE		Any locale related exception
EC-LOCALE-IMP	Imp	Implementor-defined locale related exception
EC-LOCALE-INCOMPATIBLE	Fatal	The referenced locale does not specify the expected characters in LC_COLLATE
EC-LOCALE-INVALID	Fatal	Locale content is invalid or incomplete
EC-LOCALE-INVALID-PTR	Fatal	Pointer does not reference a saved locale
EC-LOCALE-MISSING	Fatal	The specified locale is not available
EC-LOCALE-SIZE	Fatal	Digits were truncated in locale editing
EC-OO		Any predefined OO related exception
EC-OO-CONFORMANCE	Fatal	Failure for an object-view

Table 14 — Exception-names and exception conditions (Continued)

Exception-name	Cat	Description
EC-OO-EXCEPTION	Fatal	An exception object was not handled
EC-OO-IMP	Imp	Implementor-defined OO exception
EC-OO-METHOD	Fatal	Requested method is not available
EC-OO-NULL	Fatal	Method invocation was attempted with a null object reference
EC-OO-RESOURCE	Fatal	Insufficient system resources to create the object
EC-OO-UNIVERSAL	Fatal	A runtime type check failed
EC-ORDER		Ordering exception
EC-ORDER-IMP	Imp	Implementor-defined ordering exception
EC-ORDER-NOT-SUPPORTED	Fatal	ISO/IEC 14651 ordering table or ordering level not supported
EC-OVERFLOW		Overflow condition
EC-OVERFLOW-IMP	Imp	Implementor-defined overflow condition
EC-OVERFLOW-STRING	NF	STRING overflow condition
EC-OVERFLOW-UNSTRING	NF	UNSTRING overflow condition
EC-PROGRAM		Inter-program communication exception
EC-PROGRAM-ARG-MISMATCH	Fatal	Parameter mismatch
EC-PROGRAM-ARG-OMITTED	Fatal	A reference to an omitted argument
EC-PROGRAM-CANCEL-ACTIVE	Fatal	Canceled program active
EC-PROGRAM-IMP	Imp	Implementor-defined inter-program communication exception
EC-PROGRAM-NOT-FOUND	Fatal	Called program not found
EC-PROGRAM-PTR-NULL	Fatal	Program-pointer used in CALL is set to NULL
EC-PROGRAM-RECURSIVE-CALL	Fatal	Called program active
EC-PROGRAM-RESOURCES	Fatal	Resources not available for called program
EC-RAISING		EXIT ... RAISING or GOBACK RAISING exception
EC-RAISING-IMP	Imp	Implementor-defined EXIT ... RAISING or GOBACK RAISING exception
EC-RAISING-NOT-SPECIFIED	Fatal	EXIT ... RAISING or GOBACK RAISING an EC-USER exception condition not specified in RAISING phrase of procedure division header
EC-RANGE		Range exception
EC-RANGE-IMP	Imp	Implementor-defined range exception
EC-RANGE-INDEX	Fatal	Index set outside the range of values allowed by the implementor
EC-RANGE-INSPECT-SIZE	Fatal	Size of replace items in INSPECT differs
EC-RANGE-INVALID	NF	Starting value of THROUGH range greater than ending value

Table 14 — Exception-names and exception conditions (Continued)

Exception-name	Cat	Description
EC-RANGE-PERFORM-VARYING	Fatal	Setting of varied item in PERFORM is negative
EC-RANGE-PTR	Fatal	Pointer SET UP or DOWN is outside range
EC-RANGE-SEARCH-INDEX	NF	No table entry found in SEARCH because initial index out of range
EC-RANGE-SEARCH-NO-MATCH	NF	No table entry found in SEARCH because no entry matched criteria
EC-REPORT		Report writer exception
EC-REPORT-ACTIVE	Fatal	INITIATE on an active report
EC-REPORT-COLUMN-OVERLAP	NF	Overlapping report items
EC-REPORT-FILE-MODE	Fatal	An INITIATE statement was executed for a file connector that was not open in the extend or output mode
EC-REPORT-IMP	Imp	Implementor-defined report writer exception
EC-REPORT-INACTIVE	Fatal	GENERATE or TERMINATE on an inactive report
EC-REPORT-LINE-OVERLAP	NF	Overlapping report lines
EC-REPORT-NOT-TERMINATED	NF	Report file closed with active report
EC-REPORT-PAGE-LIMIT	NF	Vertical page limit exceeded
EC-REPORT-PAGE-WIDTH	NF	Page width exceeded
EC-REPORT-SUM-SIZE	Fatal	Overflow of sum counter
EC-REPORT-VARYING	Fatal	VARYING clause expression noninteger
EC-SCREEN		Screen handling exception
EC-SCREEN-FIELD-OVERLAP	NF	Screen fields overlap
EC-SCREEN-IMP	Imp	Implementor-defined screen handling exception
EC-SCREEN-ITEM-TRUNCATED	NF	Screen field too long for line
EC-SCREEN-LINE-NUMBER	NF	Screen item line number exceeds terminal size
EC-SCREEN-STARTING-COLUMN	NF	Screen item starting column exceeds line size
EC-SIZE		Size error exception
EC-SIZE-ADDRESS	Fatal	Invalid pointer arithmetic
EC-SIZE-EXPONENTIATION	Fatal	Exponentiation rules violated
EC-SIZE-IMP	Imp	Implementor-defined size error exception
EC-SIZE-OVERFLOW	Fatal	Arithmetic overflow in calculation
EC-SIZE-TRUNCATION	Fatal	Significant digits truncated in store
EC-SIZE-UNDERFLOW	Fatal	Floating-point underflow
EC-SIZE-ZERO-DIVIDE	Fatal	Division by zero
EC-SORT-MERGE		SORT or MERGE exception
EC-SORT-MERGE-ACTIVE	Fatal	File SORT or MERGE executed when one is already active

Table 14 — Exception-names and exception conditions (Continued)

Exception-name	Cat	Description
EC-SORT-MERGE-FILE-OPEN	Fatal	A USING or GIVING file is open upon execution of a SORT or MERGE
EC-SORT-MERGE-IMP	Imp	Implementor-defined SORT or MERGE exception
EC-SORT-MERGE-RELEASE	Fatal	RELEASE record too long or too short
EC-SORT-MERGE-RETURN	Fatal	RETURN executed when at end condition exists
EC-SORT-MERGE-SEQUENCE	Fatal	Sequence error on MERGE USING file
EC-STORAGE		Storage allocation exception
EC-STORAGE-IMP	Imp	Implementor-defined storage allocation exception
EC-STORAGE-NOT-ALLOC	NF	The data-pointer specified in a FREE statement does not identify currently allocated storage
EC-STORAGE-NOT-AVAIL	NF	The amount of storage requested by an ALLOCATE statement is not available
EC-USER		User-defined exception condition
EC-USER-*suffix* (user specifies *suffix*)	NF	Level-3 user-defined exception condition
EC-VALIDATE		VALIDATE exception
EC-VALIDATE-CONTENT	NF	VALIDATE content error
EC-VALIDATE-FORMAT	NF	VALIDATE format error
EC-VALIDATE-IMP	Imp	Implementor-defined VALIDATE exception
EC-VALIDATE-RELATION	NF	VALIDATE relation error
EC-VALIDATE-VARYING	Fatal	VARYING clause expression noninteger

14.6.12.2 Incompatible data

Incompatible data exists when the content of a sending operand is not valid only in the following cases:

1) When the content of a boolean sending item is referenced during the execution of a statement and the content of that sending operand would evaluate to false in a boolean class condition, the result of the reference is undefined and an EC-DATA-INCOMPATIBLE exception condition is set to exist, except in the following circumstances:

   — a sending item is referenced in a class condition, or
   — a sending item is processed in a VALIDATE statement.

   The EC-DATA-INCOMPATIBLE exception condition is set to exist for a class condition and a VALIDATE statement when invalid data is detected during item identification.

   NOTE For example, a subscript reference during a class test could cause the EC-DATA-INCOMPATIBLE exception condition to exist.

2) When the content of a numeric sending item that is not described with an IEEE floating-point usage is referenced during the execution of a statement and the content of that sending operand would evaluate to false in a numeric class condition, the result of the reference is undefined and an EC-DATA-INCOMPATIBLE exception condition is set to exist, except in the following circumstances:

   — a sending item is referenced in a class condition, or

— a sending item is processed in a VALIDATE statement.

The EC-DATA-INCOMPATIBLE exception condition is set to exist for a class condition and a VALIDATE statement when invalid data is detected during item identification.

NOTE For example, a subscript reference during a class test could cause the EC-DATA-INCOMPATIBLE exception condition to exist.

3) When a sending operand is described with an IEEE floating-point usage, and the content of the sending operand would evaluate to true in an infinity class condition or to true in a represents-not-a-number class condition, the EC-DATA-NOT-FINITE exception condition is set to exist, except in the following circumstances:

— a sending item is referenced in a class condition, or
— a sending item is referenced in a sign condition, or
— the sending and receiving items in a simple format MOVE statement are both of IEEE floating-point usages, or
— a sending item is processed in a VALIDATE statement.

The EC-DATA-INCOMPATIBLE exception condition is set to exist for a class condition and a VALIDATE statement when invalid data is detected during item identification.

NOTE For example, a subscript reference during a class test could cause the EC-DATA-INCOMPATIBLE exception condition to exist.

4) When a sending operand is described with an IEEE decimal floating-point usage, and the content of the sending operand would evaluate to false in a numeric class condition, to false in an infinity class condition, and to false in a represents-not-a-number class condition, the EC-DATA-NOT-DECIMAL-ENCODING exception condition is set to exist, except in the following circumstances:

— a sending item is referenced in a class condition, or
— a sending item is referenced in a sign condition, or
— the sending and receiving items in a simple format MOVE statement are both of IEEE floating-point usages, or
— a sending item is processed in a VALIDATE statement.

The EC-DATA-INCOMPATIBLE exception condition is set to exist for a class condition and a VALIDATE statement when invalid data is detected during item identification.

NOTE For example, a subscript reference during a class test could cause the EC-DATA-INCOMPATIBLE exception condition to exist.

5) When a numeric-edited data item is the sending operand of a de-editing MOVE statement and the content of that data item is not a possible result for any editing operation in that data item, the result of the MOVE operation is undefined and an EC-DATA-INCOMPATIBLE exception condition is set to exist.

6) When the internal format of an any-length elementary item is not correctly formed or does not agree with the corresponding ANY LENGTH clause an EC-DATA-INCOMPATIBLE exception condition is set to exist.

7) When the internal format of a dynamic-capacity table, as defined by the implementor, is not correctly formed or does not agree with the corresponding OCCURS clause an EC-DATA-INCOMPATIBLE exception condition is set to exist.

If the content of a sending operand is not referenced by a given execution of a statement, any incompatible data in that operand is not detected. If part of a sending operand's content is referenced by a given execution of a statement, it is undefined whether any incompatible data in the unreferenced content is detected.

NOTE The content of a data item is not referenced when the data item is a receiving operand, unless that data item is also a sending data item.

14.7 Common phrases and features for statements

This clause provides a description of the common phrases and features that pertain to or appear in several different statements.

14.7.1 At end condition

The at end condition is associated with a sort-merge file or the I-O status for a file connector. For sort-merge files, the at end condition is set to exist when the sort or merge operation has returned all of the records that were sent to it and there are no more records to be sorted or merged. It no longer exists when the execution of the SORT or MERGE operation referencing the sort-merge file terminates. For other files, the at end condition exists when the first character of the I-O status value for the associated file connector is a '1'.

14.7.2 Invalid key condition

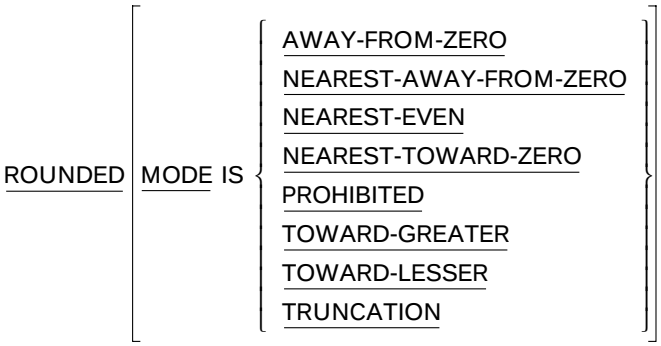
The invalid key condition is associated with the I-O status for file connectors that are not associated with a sort-merge file. It exists when the first character of the I-O status value for the associated file connector is a '2'.

14.7.3 ROUNDED phrase

If, after decimal point alignment, the number of places in the fractional part of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant identifier, truncation is relative to the size provided for the resultant identifier.

When the low-order integer positions in a resultant identifier are represented by the symbol P in the picture character-string for that resultant identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

14.7.3.1 General format



14.7.3.2 General rules

1) If the MODE phrase is not specified, the form of rounding is as described in 11.9.5, DEFAULT ROUNDED clause.

2) If the ROUNDED phrase is not specified, execution is as if ROUNDED MODE IS TRUNCATION had been specified.

3) If the AWAY-FROM-ZERO phrase is specified and the arithmetic value cannot be exactly represented in the resultant identifier, the arithmetic value is rounded to the nearest value with a larger magnitude that can be represented in the resultant identifier.

4) If the NEAREST-AWAY-FROM-ZERO phrase is specified or implied and the arithmetic value cannot be exactly represented in the resultant identifier, the arithmetic value is rounded to the nearest value that can be represented in the resultant identifier. If two such values are equally near, the value whose magnitude is larger is chosen.

5) If the NEAREST-EVEN phrase is specified and the arithmetic value cannot be exactly represented in the resultant identifier, the arithmetic value is rounded to the nearest value that can be represented in the resultant identifier. If two such values are equally near, the value whose rightmost digit is even is chosen.

   NOTE This method is sometimes known as 'banker's rounding'.

6) If the NEAREST-TOWARD-ZERO phrase is specified and the arithmetic value cannot be exactly represented in the resultant identifier, the arithmetic value is rounded to the nearest value that can be represented in the resultant identifier. If two such values are equally near, the value whose magnitude is smaller is chosen.

7) If the PROHIBITED phrase is specified, and the arithmetic value cannot be represented exactly in the resultant identifier, the EC-SIZE-TRUNCATION exception condition is set to exist and the results of the operation are undefined.

8) If the TOWARD-GREATER phrase is specified, and the arithmetic value cannot be represented exactly in the resultant identifier, the arithmetic value is rounded to the nearest larger value that can be represented in the resultant identifier.

9) If the TOWARD-LESS phrase is specified, and the arithmetic value cannot be represented exactly in the resultant identifier, the arithmetic value is rounded to the nearest smaller value that can be represented in the resultant identifier.

10) If the TRUNCATION phrase is specified or implied, and the arithmetic value cannot be represented exactly in the resultant identifier, the arithmetic value is rounded to the nearest value with a smaller magnitude that can be represented in the resultant identifier.

14.7.4 SIZE ERROR phrase and size error condition

The size error condition may occur as a result of the execution of an ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statement or of the evaluation of an arithmetic expression. Checking of a size error condition may be enabled by:

— specifying the SIZE ERROR phrase of an arithmetic statement, or by

— using a TURN directive to turn on checking of the EC-SIZE exception condition.

When the SIZE ERROR phrase is specified for an arithmetic statement, checking for the size error condition is enabled for the arithmetic operations that take place in developing and storing the result of that arithmetic statement. Any exception condition, including EC-SIZE, that is raised during item identification for the operands used during the execution of the arithmetic statement is processed as defined for that exception condition and execution of the arithmetic statement ceases. Execution resumes as indicated for that exception condition. If an EC-SIZE exception condition exists during the execution of the arithmetic statement other than during item identification and a SIZE ERROR phrase is specified for the statement, processing of the size error condition occurs as described below for the SIZE ERROR phrase, and no EC-SIZE exception declaratives are performed.

The size error condition exists in the following cases:

1) if the rules for evaluation of exponentiation are violated;

2) if the divisor in a divide operation or in a DIVIDE statement is zero;

3) if, after radix point alignment and any rounding specified by the ROUNDED phrase, the absolute value of the result of an arithmetic statement exceeds the largest value that may be contained in the associated resultant data item;

4) if standard arithmetic is specified for the source unit or the implementor defines that the range of values allowed for the intermediate data item is to be checked, when an arithmetic operation on the intermediate data item would cause the new value to be outside of the allowed range;

5) If standard-binary arithmetic is in effect, when an arithmetic operation on the intermediate data item would cause the new value to be outside of the range allowed for a standard-binary intermediate data item.

6) If standard-decimal arithmetic is in effect, when an arithmetic operation on the intermediate data item would cause the new value to be outside of the range allowed for a standard-decimal intermediate data item.

7) if the rules for a statement or an expression explicitly specify that an EC-SIZE exception condition is set to exist.

If the size error condition exists and the SIZE ERROR phrase is specified, the following occurs:

1) If the size error condition occurred during the arithmetic operations specified by the arithmetic statement, the values of all of the resultant data items remain unchanged from the values they had at the start of the execution of the arithmetic statement. Execution proceeds as indicated in rule 3, below.

2) If the absolute value of the result of the arithmetic operation exceeds the maximum value allowed for any resultant identifier, the content of that resultant identifier is not changed from the content that existed at the start of the execution of the arithmetic statement. The values of resultant identifiers for which the size error condition did not occur are the same as they would have been if the size error condition had not existed for any of the resultant identifiers. Execution proceeds as indicated in rule 3, below.

3) After completion of the arithmetic operations, and possibly the storing of values into resultant data items as specified in rule 2, control is transferred to the imperative-statement specified in the SIZE ERROR phrase and execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the SIZE ERROR phrase, control is transferred to the end of the arithmetic statement and the NOT SIZE ERROR phrase, if specified, is ignored.

If the size error condition exists and a SIZE ERROR phrase is not specified:

1) if the rules for evaluation of exponentiation are violated, the EC-SIZE-EXPONENTIATION exception condition is set to exist,

2) if the divisor in a divide operation or the DIVIDE statement is zero, the EC-SIZE-ZERO-DIVIDE exception condition is set to exist,

3) if the absolute value of the result of an arithmetic statement exceeds the largest value that may be contained in the associated resultant data item, the EC-SIZE-TRUNCATION exception condition is set to exist,

4) If an arithmetic operation on a standard intermediate data item of the form corresponding to the mode of arithmetic currently in effect would cause the new value to be outside of the allowed range, the applicable EC-SIZE exception condition, either EC-SIZE-OVERFLOW or EC-SIZE-UNDERFLOW, is set to exist,

and processing proceeds as specified in 14.6.12.1.2, Fatal exception conditions. If a NOT SIZE ERROR phrase is specified, it is ignored.

If no size error condition occurs during the execution of the arithmetic operations specified by an arithmetic statement or expression or while storing into the resultant identifiers, the SIZE ERROR phrase, if specified, is ignored and control is transferred to the end of the arithmetic statement or expression or to the imperative-statement specified in the NOT SIZE ERROR phrase if it is specified. In the latter case, execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the NOT SIZE ERROR phrase, control is transferred to the end of the arithmetic statement.

14.7.5 CORRESPONDING phrase

For the purpose of this discussion, D1 and D2 are identifiers that refer to alphanumeric group items, bit group items, national group items, strongly-typed group items, or variable-length groups.

NOTE When D1 and D2 refer to national groups or bit groups, D1 and D2 are processed as group items and not as elementary items.

A pair of data items correspond if:

1) A data item in D1 and a data item in D2 are not implicitly or explicitly described with the keyword FILLER and have the same data-name and the same qualifiers, if any, up to, but not including, D1 and D2.

2) In a MOVE statement, at least one of the data items is an elementary data item and the resulting move is valid according to the rules for the MOVE statement.

3) In an ADD or SUBTRACT statement, both of the data items are numeric data items.

4) Neither data item contains an OCCURS, REDEFINES, or RENAMES clause or is of class index, object, or pointer.

5) Neither data item is subordinate to a group item that is subordinate to D1 or D2 when the group item contains an OCCURS or REDEFINES clause.

6) The name of each data item that satisfies the above conditions is unique after application of the implied qualifiers.

Any item identification associated with a corresponding pair of operands is done at the start of the execution of the statement containing the CORRESPONDING phrase, not at the start of the implied statement used for the pair of operands. The implied statements are executed in the order in which the elements in the group data item immediately following CORRESPONDING are specified.

For the arithmetic statements with the CORRESPONDING phrase, if the SIZE ERROR phrase is specified and one or more of the implied statements raises a size error condition, the imperative-statement in the SIZE ERROR phrase is executed after all of the implied statements are completed and the NOT SIZE ERROR phrase, if specified, is ignored.

For the arithmetic statements with the CORRESPONDING phrase, if the SIZE ERROR phrase is not specified and any of the implied statements raises a size error condition, the level-3 EC-SIZE exception condition for the last of the implied statements that raised a size error condition is set to exist after all of the implied statements are completed.

For any statement with the CORRESPONDING phrase, if any of the implied statements would set the EC-DATA-INCOMPATIBLE exception condition to exist, the EC-DATA-INCOMPATIBLE exception condition is set to exist after all of the implied statements are completed.

14.7.6 Arithmetic statements

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common features.

1) The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.

2) For ADD, DIVIDE, MULTIPLY, and SUBTRACT statements when native arithmetic is in effect:

   a) When none of the operands is an intrinsic function or a data item described with usage binary-char, binary-short, binary-long, binary-double, float-short, float-long, or float-extended, or of an IEEE floating-point usage, the composite of operands shall not contain more than 31 digits.

b) When any of the operands is an intrinsic function or a data item described with usage binary-char, binary-short, binary-long, binary-double, float-short, float-long, or float-extended, or is of an IEEE floating-point usage, the composite of the operands that are not intrinsic functions or data items described with usage binary-char, binary-short, binary-long, binary-double, float-short, float-long, or float-extended, or are not of an IEEE floating-point usage shall not contain more than 31 digits.

The composite of operands is a hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points.

3) When standard, standard-binary or standard-decimal arithmetic is in effect, each arithmetic statement is defined in terms of one or more arithmetic expressions. Storing into a resultant-identifier from an intermediate form shall be according to the rules for the MOVE statement. The ROUNDED phrase applies only to this move.

NOTE When standard arithmetic is in effect, a size error condition may occur during the execution of arithmetic operations used to compute the result as well as during the final move.

4) The arithmetic statements may have single or multiple resultant identifiers. These statements have common rules for storing in the resultant identifiers. The execution of these statements proceeds in the following order:

a) The initial evaluation of the statement is done and the result of this operation is placed in an intermediate data item. If standard arithmetic is in effect, a standard intermediate data item is used. Otherwise, an implementor-defined intermediate data item is used. The rules indicating which data items or literals are part of this evaluation are given in the rules for the individual statements. All item identification for the data items involved in the initial evaluation is done at the start of the execution of the statement. If the size error condition is raised during the initial evaluation, none of the resultant data items are changed and execution proceeds as indicated in 14.7.4, SIZE ERROR phrase and size error condition.

b) If the size error condition was not raised during the initial evaluation, the intermediate data item is stored in or combined with and then stored in each single resulting data item in the left-to-right order in which the receiving data items are specified in the statement. Item identification for the receiving data items is done as each data item is accessed unless it was already done in step a. If the size error condition is raised when attempting to store in a resulting data item, only that data item remains unchanged and processing proceeds to the next resulting data item to the right.

NOTE Because rule 4 specifies the use of an intermediate data item to hold the value resulting from the initial arithmetic evaluation, and that arithmetic evaluation is complete before any data is moved from that intermediate data item to any receiving operands, the results contained in the receiving operands for all arithmetic statements are defined regardless of whether any of the sending operands share storage with any of the receiving operands. Those results in the receiving operands are the same as if no receiving operand shared any part of its storage area with any sending operand.

14.7.7 THROUGH phrase

This specification applies to THROUGH phrases specified in the VALUE clause and the EVALUATE statement.

A THROUGH phrase specifies a range of values, literal-1 through literal-2. The set of values included in the range is determined by the following rules:

1) When the range of values is defined by numeric literals, the range of values includes literal-1, literal-2, and all algebraic values between literal-1 and literal-2.

2) When the range of values is defined by alphanumeric or national literals, the range of values depends on the collating sequence used for evaluation of the range.

When there is no IN alphabet-name phrase specified for the range, the collating sequence is defined by the implementor.

NOTE The intent is to allow an implementor to define the collating sequence in a manner compatible with previous COBOL implementations.

When the IN alphabet-name phrase is specified, the collating sequence used for range evaluation is the collating sequence defined by that alphabet. When the alphabet-name is associated with a locale, the range of values is determined from locale category LC_COLLATE in the specific locale associated with that alphabet-name or, if none, in the current locale. When the alphabet-name is associated with the native collating sequence, the computer's runtime collating sequence is used at the time the values are used in execution.

The range of values includes all values collating at the starting value and all successive ascending values in the applicable collating sequence up to and including all values collating at the ending value.

When the value of literal-1 is greater than the value of literal-2 in the collating sequence in effect at runtime, the EC-RANGE-INVALID exception condition is set to exist, and, upon completion of any exception processing, execution proceeds as if the range of values were empty.

14.7.8 RETRY phrase

The RETRY phrase is specified in an input-output statement to indicate whether the MSCS is to continue to attempt to obtain access in the event that a file or record is locked.

14.7.8.1 General format

```
RETRY { arithmetic-expression-1 TIMES          }
      { FOR arithmetic-expression-2 SECONDS    }
      { FOREVER                                }
```

14.7.8.2 General rules

1) Arithmetic-expression-1 specifies the number of times after the initial failure that the MSCS shall attempt to gain access to the locked resource and complete the requested input-output operation. The implementor determines the interval between these attempts. If arithmetic-expression-1 does not evaluate to an integer, the value of arithmetic-expression-1 is rounded up to the next whole number.

2) Arithmetic-expression-2 specifies the number of seconds in the timeout period. The I-O statement behaves as though the length of the timeout period were stored in a temporary data item whose picture is 9(n)V9(m), in the manner specified by this rule. The implementor shall specify the value of m, which may be zero, and the value of n, which shall be greater than zero. The implementor shall specify the maximum meaningful value of arithmetic-expression-2. If arithmetic-expression-2 is greater than this maximum meaningful value, the maximum meaningful value is placed into the temporary data item; otherwise, arithmetic-expression-2 is used as the sending item and the temporary data item as the receiving item in an implicit COMPUTE statement without the ROUNDED phrase. During the timeout period, the MSCS shall attempt to gain access to the locked resource and complete the requested input-output operation. The implementor shall specify the techniques used to determine the frequency of retries during the timeout period.

3) If the FOREVER phrase is specified, the MSCS shall attempt to gain access to a locked resource until the input-output operation has been completed.

4) If the I/O operation is unsuccessful on the first attempt because of a file sharing conflict condition or a record operation conflict condition, the following apply:

   a) If the RETRY phrase is not specified or the result of the evaluation of arithmetic-expression-1 or arithmetic-expression-2 is negative or zero, the statement is unsuccessful, the appropriate value is placed in the I-O status associated with the file connector, and execution proceeds as indicated for unsuccessful execution in the applicable statement; otherwise,

   b) The MSCS attempts to complete the input-output operation as specified in general rules 1, 2, or 3.

If the MSCS permits the requested access on one of these attempts, the statement is successful and the results are as if the file sharing or record operation conflict had never occurred.

Otherwise, the statement is unsuccessful, the appropriate value is placed in the I-O status associated with the file connector, and execution proceeds as indicated for unsuccessful execution in the applicable statement.

14.8 Conformance for parameters and returning items

The conformance rules for parameters and returning items apply at compile time when an explicit reference is made to them from a syntax rule.

NOTE 1 Conformance rules for parameters and returning items are checked at compile time for:

— an INVOKE statement on an object reference that is not a universal object reference,
— a reference to a user-defined function,
— a program-prototype format of the CALL statement.

The conformance rules for parameters and returning items apply at runtime when an explicit reference is made to them from a general rule.

NOTE 2 Conformance rules for parameters and returning items are checked at runtime for:

— a universal object reference, if exception condition EC-OO-UNIVERSAL is enabled
— the on-overflow and on-exception formats of the CALL statement, if exception condition EC-PROGRAM-ARG-MISMATCH is enabled.

NOTE 3 Conformance rules for parameters and returning items are checked for an object-view at compile time. Additional conformance rules for the object referenced by the object view are given in the rules of object-view; these are checked at runtime if exception condition EC-OO-CONFORMANCE is enabled.

14.8.1 Parameters

The number of arguments in the activating element shall be equal to the number of formal parameters in the activated element, with the exception of trailing formal parameters that are specified with an OPTIONAL phrase in the procedure division header of the activated element and omitted from the list of arguments of the activating element.

If both an argument and its corresponding formal parameters are elementary items, the conformance rules for elementary items apply; otherwise, the conformance rules for group items apply.

NOTE A bit group or national group is treated as an elementary item.

14.8.1.1 Group items

If either the formal parameter or the argument is an alphanumeric group item and neither of them is strongly typed:

1) If the argument is passed by reference, that argument or the formal parameter corresponding to that argument shall be an alphanumeric group item or an elementary item of category alphanumeric, and the formal parameter shall be described with the same number or a smaller number of bytes as the corresponding argument.

2) If the argument is passed by content, the conformance rules are the same as for a MOVE statement with the argument as the sending operand and the corresponding formal parameter as the receiving operand.

NOTE If an argument is a group with a level number other than 1 and its subordinate items are described such that the implementation inserts slack bits or bytes, the alignment of the subordinate elementary items might not correspond between the argument and the formal parameter.

If either the formal parameter or the corresponding argument is a strongly-typed group item, both shall be of the same type.

For an argument or formal parameter that is described as an occurs-depending group item passed by reference, the maximum length is used. For an occurs-depending group item passed by content, the length of the argument is determined by the rules of the OCCURS clause for a sending data item.

14.8.1.2 Elementary items

The conformance rules for elementary items depend on whether the argument is passed by reference, by content, or by value.

14.8.1.2.1 Elementary items passed by reference

If either the formal parameter or the corresponding argument is an object reference, the corresponding argument or formal parameter shall be an object reference following these rules:

1) If either the argument or the formal parameter is a universal object reference, the corresponding formal parameter or argument shall be a universal object reference.

2) If either the argument or the formal parameter is described with an interface-name, the corresponding formal parameter or argument shall be described with the same interface-name.

3) If either the argument or the formal parameter is described with a class-name, the corresponding formal parameter or argument shall be described with the same class-name, and the FACTORY and ONLY phrases shall be the same.

4) If the formal parameter is described with the ACTIVE-CLASS phrase, one of the following conditions shall be true:

    a) The argument shall be an object reference described with the ACTIVE-CLASS phrase, where the presence or absence of the FACTORY phrase is the same as in the formal parameter, and the method to be activated shall be invoked with the predefined object references SELF or SUPER, or with an object reference described with the ACTIVE-CLASS phrase.

    b) The argument shall be an object reference described with a class-name and the ONLY phrase, where the presence or absence of the FACTORY phrase is the same as in the formal parameter, and the method to be activated shall be invoked with that class-name or with an object reference described with that class-name and the ONLY phrase.

If either the argument or the formal parameter is of class pointer, the corresponding formal parameter or argument shall be of class pointer and the corresponding items shall be of the same category. If either is a restricted pointer, both shall be restricted and of the same type.

If neither the formal parameter nor the argument is of class object or pointer, the conformance rules are the following:

1) If the activated element is a program for which there is no program-specifier in the REPOSITORY paragraph of the activating element and there is no NESTED phrase specified on the CALL statement, the formal parameter shall be of the same length as the corresponding argument.

2) If the activated element is one of the following:

    — a program for which there is a program-specifier in the REPOSITORY paragraph of the activating element
    — a program and the NESTED phrase is specified on the CALL statement
    — a method
    — a function

then the definition of the formal parameter and the definition of the argument shall have the same BLANK WHEN ZERO, JUSTIFIED, PICTURE, USAGE, and SIGN clauses, with the following exceptions:

a) Currency symbols match if and only if the corresponding currency strings are the same.

b) Period picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both the activating and the activated runtime elements or for neither of them. Comma picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both the activating and the activated runtime elements or for neither of them.

Additionally:

a) Locale specifications in the PICTURE clauses match if and only if:

— both specify the same SIZE phrase in the LOCALE phrase of the PICTURE clause, and

— both specify the LOCALE phrase without a locale-name or both specify the LOCALE phrase with the same external identification, where the external identification is the external-locale-name or literal value associated with a locale-name in the LOCALE clause of the SPECIAL-NAMES paragraph.

b) A bit group item matches an elementary bit data item described with the same number of boolean positions.

c) A national group item matches an elementary data item of usage national described with the same number of national character positions.

d) If the formal parameter is described with the ANY LENGTH clause, its length is considered to match the length of the corresponding argument.

e) If the argument is described with the ANY LENGTH clause, the corresponding formal parameter shall be described with the ANY LENGTH clause.

14.8.1.2.2 Elementary items passed by content or by value

If the formal parameter is an object reference described with the ACTIVE-CLASS phrase, one of the following conditions shall be true:

1) The method to be activated shall be invoked with the predefined object references SELF or SUPER, or with an object reference described with the ACTIVE-CLASS phrase, and a SET statement shall be valid in the activating unit with the argument as the sending operand and an object reference described with the ACTIVE-CLASS phrase, where the presence or absence of the FACTORY phrase is the same as in the formal parameter, as the receiving operand.

2) The method to be activated shall be invoked with a class-name or with an object reference described with a class-name and the ONLY phrase, and a SET statement shall be valid in the activating unit with the argument as the sending operand and an object reference described with that class-name and the ONLY phrase, where the presence or absence of the FACTORY phrase is the same as in the formal parameter, as the receiving operand.

If the formal parameter is of class pointer or an object reference described without the ACTIVE-CLASS phrase, the conformance rules shall be the same as if a SET statement were performed in the activating runtime element with the argument as the sending operand and the corresponding formal parameter as the receiving operand.

If the formal parameter is not of class object or pointer, the conformance rules are the following:

1) If the activated element is a program for which there is no program-specifier in the REPOSITORY paragraph of the activating element and there is no NESTED phrase specified on the CALL statement, the formal parameter shall be of the same length as the corresponding argument.

2) If the activated element is one of the following:

   — a program for which there is a program-specifier in the REPOSITORY paragraph of the activating element
   — a program and the NESTED phrase is specified on the CALL statement
   — a method
   — a function

   then the conformance rules depend on the type of the formal parameter as specified in the following rules:

   a) If the formal parameter is numeric, the conformance rules are the same as for a COMPUTE statement with the argument as the sending operand and the corresponding formal parameter as the receiving operand.

   b) If the formal parameter is an index data item, the conformance rules are the same as for a SET statement with the argument as the sending operand and the corresponding formal parameter as the receiving operand.

   c) If the formal parameter is described with the ANY LENGTH clause, its length is considered to match the length of the corresponding argument.

   d) Otherwise, the conformance rules are the same as for a MOVE statement with the argument as the sending operand and the corresponding formal parameter as the receiving operand.

14.8.2 Returning items

A returning item shall be specified in the activating statement if and only if a returning item is specified in the procedure division header of the activated element. A returning item is implicitly specified in the activating element when a function or inline method invocation is referenced.

The returning item in the activated element is the sending operand, the corresponding returning item in the activating element is the receiving operand.

The rules for conformance between the sending operand and the receiving operand depend on whether at least one of the operands is an alphanumeric group item or both operands are elementary items.

NOTE A bit group or national group is treated as an elementary item.

14.8.2.1 Group items

If either the sending or the receiving operand is an alphanumeric group item, and neither of them is strongly typed, the corresponding returning item shall be an alphanumeric group item or an elementary item of category alphanumeric, and the receiving operand shall be of the same length as the sending operand.

NOTE If a returning item in an activating element is a group with a level number other than 1 and its subordinate items are described such that the implementation inserts slack bits or bytes, the alignment of the subordinate elementary items might not correspond between the returning item in the activating runtime element and the returning item in the activated runtime element.

If either of the operands is a strongly-typed group item, both shall be of the same type.

For an operand that is described as a variable-occurrence data item, the maximum length is used.

14.8.2.2 Elementary items

If either of the operands is an object reference, the corresponding item shall be an object reference, and the following rules apply:

1) If the returning item in the activated element is not described with an ACTIVE-CLASS phrase, the conformance rules are the same as if a SET statement were performed in the activated runtime element with the returning

item in the activated element as the sending operand and the corresponding returning item in the activating element as the receiving operand.

2) If the returning item in the activated element is described with an ACTIVE-CLASS phrase, the conformance rules are the same as the conformance rules for a SET statement specified in the activating element with the following operands:

   a) A receiving operand that is the returning item in the activating element.

   b) A sending operand that is an object reference described as follows:

      1. If the activated method is invoked with a class-name, the sending object reference is described with that same class-name and an ONLY phrase.

      2. If the activated method is invoked with the predefined object reference SELF or SUPER, the sending object reference is described with an ACTIVE-CLASS phrase.

      3. If the activated method is invoked with an object reference that is described with an interface-name, the sending object reference is a universal object reference.

      4. If the activated method is invoked with any other object reference, the sending operand has the same description as that object reference.

   If the sending operand defined above is described with a class-name or an ACTIVE-CLASS phrase, the presence or absence of the FACTORY phrase is the same as in the returning item of the activated element.

If the sending operand is not an object reference, the receiving operand shall have the same BLANK WHEN ZERO, JUSTIFIED, PICTURE, USAGE, and SIGN clauses, with the following exceptions:

1) Currency symbols match if and only if the corresponding currency strings are the same.

2) Period picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both the activating and the activated runtime elements or for neither of them.

3) Comma picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both the activating and the activated runtime elements or for neither of them.

Additionally, if the sending operand is not an object reference:

1) Locale specifications in the PICTURE clauses match if and only if:

   — both specify the same SIZE phrase in the LOCALE phrase of the PICTURE clause, and

   — both specify the LOCALE phrase without a locale-name or both specify the LOCALE phrase with the same external identification, where the external identification is the external-locale-name or literal value associated with a locale-name in the LOCALE clause of the SPECIAL-NAMES paragraph.

2) A bit group item matches an elementary boolean data item of usage bit described with the same number of boolean positions.

3) A national group item matches an elementary data item of usage national described with the same number of national character positions.

4) If the receiving operand is described with the ANY LENGTH clause, the sending operand shall also be described with the ANY LENGTH clause.

5) If the sending operand is described with the ANY LENGTH clause, the length of the sending operand is considered to match the length of the receiving operand.

14.9 Statements

14.9.1 ACCEPT statement

The execution of a device format ACCEPT statement causes information from a device to be transferred to the specified data item, where the device is a hardware or software device in the operating environment.

The execution of a temporal format ACCEPT statement causes the information requested from the operating environment to be made available to the specified data item.

The execution of a screen format ACCEPT statement causes the following sequence of events:

- Specified or default initial values are moved to the input fields of the screen.
- The screen is displayed with the specified attributes on the terminal display screen at the specified or default location.
- The cursor is positioned to the specified or default input field.
- The operator is given the opportunity to modify the elementary input screen items.
- If inconsistent data is entered by the operator, the implementation may prompt the operator to correct the data or it may set an exception condition to exist.
- The contents of the screen items that are consistent with their descriptions are moved to the specified destination fields.
- The line and column of the cursor when input terminates are placed into the data item referenced in the CURSOR clause, if any.
- Appropriate statements in the ON EXCEPTION or NOT ON EXCEPTION clauses, if any, are executed.

14.9.1.1 General format

Format 1 (device):

```
ACCEPT identifier-1 [ FROM mnemonic-name-1 ] [ END-ACCEPT ]
```

Format 2 (temporal):

```
ACCEPT identifier-2 FROM { DATE [ YYYYMMDD ] }
                         { DAY [ YYYYDDD ]   }
                         { DAY-OF-WEEK       }
                         { TIME              } [ END-ACCEPT ]
```

Format 3 (screen):

```
ACCEPT screen-name-1
[ AT { LINE NUMBER { identifier-3 }                                          } ]
     {             { integer-1   }                                           }
     { { COLUMN }                 NUMBER { identifier-4 }                    }
     { { COL    }                        { integer-2   }                    }
[ { ON EXCEPTION imperative-statement-1     } ]
  { NOT ON EXCEPTION imperative-statement-2 }
[ END-ACCEPT ]
```

14.9.1.2 Syntax rules

1) Identifier-1 shall reference neither a strongly-typed group item nor a data item of class index, object, or pointer.

2) Mnemonic-name-1 shall be specified in the SPECIAL-NAMES paragraph of the environment division and shall be associated with an implementor-defined device-name that is identified in the operating environment as a hardware or software device capable of providing data to the program.

3) Identifier-2 shall not reference a data item of class alphabetic, boolean, index, object, or pointer.

4) Screen-name-1 may reference a group item containing screen items with FROM or VALUE clauses only if the group also contains screen items with TO or USING clauses.

5) Identifier-3 and identifier-4 shall be unsigned integer data items.

6) Neither identifier-1 nor identifier-2 shall reference a variable-length group.

14.9.1.3 General rules

FORMAT 1

1) The ACCEPT statement causes the transfer of data from the device. This data replaces the content of the data item referenced by identifier-1. Any conversion of data required between the device and the data item referenced by identifier-1 is defined by the implementor.

2) The implementor shall define, for each device, the size of a data transfer.

3) If a device is capable of transferring data of the same size as the receiving data item, the transferred data is stored in the receiving data item.

4) If a device is not capable of transferring data of the same size as the receiving data item, then:

   a) If the size of the receiving data item (or of the portion of the receiving data item not yet currently occupied by transferred data) exceeds the size of the transferred data, the transferred data is stored aligned to the left in the receiving data item (or the portion of the receiving data item not yet occupied), and additional data is requested.

   b) If the size of the transferred data exceeds the size of the receiving data item (or the portion of the receiving data item not yet occupied by transferred data), only the leftmost characters of the transferred data are stored in the receiving data item (or in the portion remaining). The remaining characters of the transferred data that do not fit into the receiving data item are ignored. If identifier-1 references a zero-length item, all the characters of the transferred data are ignored.

5) The implementor shall specify the device that is used if the FROM phrase is not specified.

FORMAT 2

6) The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier-2 according to the rules for the MOVE statement. DATE, DAY, DAY-OF-WEEK, and TIME reference the current date and time provided by the system on which the ACCEPT statement is executed. DATE, DAY, DAY-OF-WEEK, and TIME are conceptual data items and, therefore, are not described in the COBOL source unit.

7) DATE without the phrase YYYYMMDD behaves as if it had been described as an unsigned elementary integer data item of usage display six digits in length, the character positions of which, numbered from left to right, are:

Character Positions	**Contents**

1-2	The two low-order digits of the year in the Gregorian calendar.
3-4	Two numeric characters of the month of the year in the range 01 through 12.
5-6	Two numeric characters of the day of the month in the range 01 through 31.

8) DATE with the phrase YYYYMMDD behaves as if it had been described as an unsigned elementary integer data item of usage display eight digits in length, the character positions of which, numbered from left to right, are:

Character Positions	**Contents**
1-4	Four numeric characters of the year in the Gregorian calendar.
5-6	Two numeric characters of the month of the year in the range 01 through 12.
7-8	Two numeric characters of the day of the month in the range 01 through 31.

9) DAY without the phrase YYYYDDD behaves as if it had been described as an unsigned elementary integer data item of usage display five digits in length, the character positions of which, numbered from left to right, are:

Character Positions	**Contents**
1-2	The two low-order digits of the year in the Gregorian calendar.
3-5	Three numeric characters of the day of the year in the range 001 through 366.

10) DAY with the phrase YYYYDDD behaves as if it had been described as an unsigned elementary integer data item of usage display seven digits in length, the character positions of which, numbered from left to right, are:

Character Positions	**Contents**
1-4	Four numeric characters of the year in the Gregorian calendar.
5-7	Three numeric characters of the day of the year in the range 001 through 366.

11) TIME is based on the elapsed time since midnight on a 24-hour clock. If the system does not have the facility to provide fractional parts of a second the value zero is returned for those parts that are not available. TIME behaves as if it had been described as an unsigned elementary integer data item of usage display eight digits in length, the characters positions of which, numbered from left to right, are:

Character Positions	**Contents**
1-2	Two numeric characters of the hours past midnight in the range 00 through 23.
3-4	Two numeric characters of the minutes past the hour in the range 00 through 59.
5-6	Two numeric characters of the seconds past the minute in the range: a) 00 through 59 when a LEAP-SECOND directive with the OFF phrase is in effect b) 00 through nn, where nn is defined by the implementor, when a LEAP-SECOND directive with the ON phrase is in effect.
7-8	Two numeric characters of the hundredths of a second past the second in the range 00 through 99. 00 is returned if the system on which the ACCEPT statement is executed does not have the facility to provide the fractional part of a second.

12) DAY-OF-WEEK behaves as if it had been described as an unsigned elementary numeric integer data item one digit in length and of usage display. In DAY-OF-WEEK, the value 1 represents Monday, 2 represents Tuesday, 3 represents Wednesday, ... , 7 represents Sunday.

FORMAT 3

13) Identifiers specified in FROM or USING clauses or literals specified in FROM or VALUE clauses provide the initial values displayed for the associated screen item during execution of an ACCEPT screen statement. For elementary screen items that have no FROM, USING, or VALUE clause, the initial value is as if a MOVE statement were executed with the screen item as the receiving field. The sending item of the MOVE statement is a figurative constant that depends on the category of the screen item as follows:

Screen item	Figurative constant
Alphabetic	Alphanumeric SPACES
Alphanumeric	Alphanumeric SPACES
Alphanumeric-edited	Alphanumeric SPACES
Boolean	ZEROS
National	National SPACES
National-edited	National SPACES
Numeric	ZEROS
Numeric-edited	ZEROS

14) Any conversion of data required between the hardware device and the data items referenced in screen-name-1 is defined by the implementor.

15) The LINE and COLUMN phrases give the position on the terminal display screen at which the screen record associated with screen-name-1 is to start. Column and line number positions are specified in terms of alphanumeric character positions. The position is relative to the leftmost character column in the topmost line of the display that is identified as column 1 of line 1. Each subordinate elementary screen item is located relative to the start of the containing screen record. Identifier-3 and identifier-4 are evaluated once at the start of execution of the statement.

16) If the LINE phrase is not specified, the screen record starts on line 1.

17) If the COLUMN phrase is not specified, the screen record starts in column 1.

18) The initial position of the cursor is determined by the CURSOR clause in the SPECIAL-NAMES paragraph.

    a) If the CURSOR clause is not specified, the initial cursor position during the execution of an ACCEPT screen statement is the start of the first input field declared within screen-name-1.

    b) If the CURSOR clause is specified, the initial cursor position is that represented by the value of the cursor locator at the beginning of the execution of the ACCEPT screen statement. If the cursor locator does not indicate a position within an input field, the cursor shall be positioned as if the CURSOR clause had not been specified.

19) During the period while the operator is able to modify each elementary screen item, each screen item is displayed on the terminal screen in accordance with any attributes specified in its screen description entry. The display may be modified as the operator selects or deselects each screen item as being the current screen item. The display of the current screen item may be modified as the operator keys data.

20) Data entered by the terminal operator in the current screen item shall be consistent with the PICTURE clause of that item. If the screen item is numeric, the entered data shall be acceptable as an argument to the NUMVAL function. If the screen item is numeric-edited, the entered data shall be acceptable as an argument to the NUMVAL-C function. It is implementor-defined when the entered data is verified. It is implementor-defined whether inconsistent data causes the EC-DATA-INCOMPATIBLE exception condition to exist or whether the system indicates an error until consistent data is entered or until execution of the ACCEPT statement is terminated.

21) If inconsistent data is entered into a screen item and allowed by the implementor to remain there, the EC-DATA-INCOMPATIBLE exception condition is set to exist. If consistent data was entered into one or more screen fields, these fields are transferred as specified in general rule 22, but the fields with inconsistent data are not transferred. The ACCEPT statement results in an unsuccessful completion, and execution proceeds as specified in general rule 25.

22) The ACCEPT screen statement causes the transfer of data from each elementary screen item that is subordinate to screen-name-1 and is specified with the TO or USING clause to the data item referenced in the TO or USING clause. For the purpose of these specifications, all such screen items are considered to be referenced by the ACCEPT screen statement. If two or more of these elementary screen items overlap, the

EC-SCREEN-FIELD-OVERLAP exception condition is set to exist and the result of the transfer of data from the elementary screen items is defined by the implementor.

The transfer occurs after the terminal operator has been given the opportunity to modify the elementary screen items and the operator has pressed a terminator key or a user-defined or context-dependent function key. This transfer occurs in the following manner:

a) If the screen item is numeric, the data is transferred as though the following statement were executed:

```
COMPUTE receiving-field = FUNCTION NUMVAL (screen-item)
```

b) If the screen item is numeric-edited, the data is transferred as though the following statement were executed:

```
COMPUTE receiving-field = FUNCTION NUMVAL-C (screen-item)
```

c) Otherwise, the data is transferred as if the following statement were executed:

```
MOVE screen-item TO receiving-field
```

where:

receiving-field is the data item referenced in the TO or USING clause, and screen-item is the screen item.

23) If the CURSOR clause is specified in the special-names paragraph, the data item referenced in the CURSOR clause shall be updated during the execution of an ACCEPT screen statement and prior to the execution of any imperative statement associated with any ON EXCEPTION or NOT ON EXCEPTION clauses for that ACCEPT statement. It shall be updated to give the line and column position of the cursor when the ACCEPT terminates.

24) If the execution of the ACCEPT statement results in a successful completion with normal termination, the ON EXCEPTION phrase, if specified, is ignored and control is transferred to the end of the ACCEPT statement or, if the NOT ON EXCEPTION phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the ACCEPT statement.

25) If the execution of the ACCEPT statement results in an unsuccessful completion, is terminated by a function key stroke, or causes an EC-SCREEN exception condition to exist, then:

a) If the ON EXCEPTION phrase is specified in the ACCEPT statement, control is transferred to imperative-statement-1. Execution then continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the ACCEPT statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

b) If the ON EXCEPTION phrase is not specified in the ACCEPT statement and an applicable Declarative exists, control is transferred to that Declarative and, if control is returned from that Declarative control is transferred to the end of the ACCEPT statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

c) If the ON EXCEPTION phrase is not specified in the ACCEPT statement and there is no applicable Declarative,

1. If the EC-DATA-INCOMPATIBLE exception condition exists, execution continues as specified in 14.6.12.1.2, Fatal exception conditions.

2. If the EC-DATA-INCOMPATIBLE exception condition does not exist, control is transferred to the end of the ACCEPT statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

14.9.2 ADD statement

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

14.9.2.1 General format

Format 1 (simple):

```
ADD { identifier-1 } ... TO { identifier-2 [ rounded-phrase ] } ...
    { literal-1    }
  [ | ON SIZE ERROR imperative-statement-1     | ]
    | NOT ON SIZE ERROR imperative-statement-2 |
  [ END-ADD ]
```

Format 2 (giving):

```
ADD { identifier-1 } ... TO { identifier-2 }
    { literal-1    }        { literal-2    }
  GIVING { identifier-3 [ rounded-phrase ] } ...
  [ | ON SIZE ERROR imperative-statement-1     | ]
    | NOT ON SIZE ERROR imperative-statement-2 |
  [ END-ADD ]
```

Format 3 (corresponding):

```
ADD { CORRESPONDING } identifier-4 TO identifier-5 [ rounded-phrase ]
    { CORR          }
  [ | ON SIZE ERROR imperative-statement-1     | ]
    | NOT ON SIZE ERROR imperative-statement-2 |
  [ END-ADD ]
```

where rounded-phrase is described in 14.7.3, ROUNDED phrase.

14.9.2.2 Syntax rules

1) When native arithmetic is in effect, the composite of operands described in 14.7.6, Arithmetic statements, is determined as follows:

   a) In format 1, by using all of the operands in the statement.

   b) In format 2, by using all of the operands in the statement excluding the data items that follow the word GIVING.

   c) In format 3, by using the two corresponding operands for each separate pair of corresponding data items.

2) Identifier-1 and identifier-2 shall reference a numeric data item.

3) Literal-1 and literal-2 shall be numeric literals.

4) Identifier-3 shall reference a numeric data item or a numeric-edited data item.

5) The words CORR and CORRESPONDING are equivalent.

6) Identifier-4 and identifier-5 shall be alphanumeric group items, national group items, variable-length groups, or strongly-typed group items and shall not be described with level-number 66.

14.9.2.3 General rules

1) When format 1 is used, the initial evaluation consists of determining the value to be added, that is literal-1 or the value of the data item referenced by identifier-1, or if more than one operand is specified, the sum of such operands. The sum of the initial evaluation and the value of the data item referenced by identifier-2 is stored as the new value of the data item referenced by identifier-2.

When standard arithmetic, standard-decimal arithmetic or standard-binary arithmetic is in effect, the result of the initial evaluation is equivalent to the result of the arithmetic expression

$$(\text{operand-}1_1 + \text{operand-}1_2 + \ldots + \text{operand-}1_n)$$

where the values of operand-1 are the values of literal-1 and the data items referenced by identifier-1 in the order in which they are specified in the ADD statement. The result of the sum of the initial evaluation and the value of the data item referenced by identifier-2 is equivalent to the result of the arithmetic expression

(initial-evaluation + identifier-2)

where initial-evaluation represents the result of the initial evaluation.

2) When format 2 is used, the initial evaluation consists of determining the sum of the operands preceding the word GIVING, that is literal-1 or the value of the data item referenced by identifier-1, and literal-2 or the value of the data item referenced by identifier-2. This value is stored as the new value of each data item referenced by identifier-3.

When standard arithmetic, standard-decimal arithmetic or standard-binary arithmetic is in effect, the result of the initial evaluation is equivalent to the result of the arithmetic expression

$$(\text{operand-}1_1 + \text{operand-}1_2 + \ldots + \text{operand-}1_n + \text{operand-}2)$$

where the values of operand-1 are the values of literal-1 and the data items referenced by identifier-1 in the order in which they are specified in the ADD statement and the value of operand-2 is the value of either literal-2 or the data item referenced by identifier-2 in the ADD statement.

3) When format 3 is used, data items in identifier-4 is added to and stored in corresponding items in identifier-5.

When standard arithmetic, standard-decimal arithmetic or standard-binary arithmetic is in effect, the result of the addition is equivalent to

(operand-1 + operand-2)

where the value of operand-1 is the value of the data item in identifier-4 and the value of operand-2 is the value of the corresponding data item in identifier-5.

4) When native arithmetic is in effect and none of the operands is described with usage binary-char, binary-short, binary-long, binary-double, float-short, float-long, or float-extended, enough places shall be carried so as not to lose any significant digits during execution.

5) Data items within identifier-4 are selected to be added to selected data items within identifier-5 according to the rules specified in 14.7.5, CORRESPONDING phrase. The results are the same as if the user had referred to each pair of corresponding identifiers in separate ADD statements.

6) Additional rules and explanations relative to this statement are given in 14.6.12.2, Incompatible data; 14.7.3, ROUNDED phrase; 14.7.4, SIZE ERROR phrase and size error condition; 14.7.5, CORRESPONDING phrase; and 14.7.6, Arithmetic statements.

14.9.3 ALLOCATE statement

The ALLOCATE statement obtains dynamic storage.

If storage is being requested for a based item, the based item is assigned the address of the obtained storage and a data-pointer, if specified, is returned containing that address.

If a specified number of characters of memory is being requested, a data-pointer addressing the obtained storage is returned.

14.9.3.1 General format

ALLOCATE { arithmetic-expression-1 CHARACTERS | data-name-1 } [INITIALIZED] [RETURNING data-name-2]

14.9.3.2 Syntax rules

1) The data item referenced by data-name-1 shall be described with the BASED clause.

2) If data-name-1 is specified, the RETURNING phrase may be omitted; otherwise, the RETURNING phrase shall be specified.

3) Data-name-2 shall reference a data item of category data-pointer.

4) If data-name-2 references a restricted data-pointer, data-name-1 shall be specified and shall reference a typed data item, and the data item referenced by data-name-2 shall be restricted to the type of data-name-1.

5) If both data-name-1 and data-name-2 are specified and data-name-1 references a strongly-typed group item, the data item referenced by data-name-2 shall be restricted to the type of data-name-1.

14.9.3.3 General rules

1) Arithmetic-expression-1 specifies a number of bytes of storage to be allocated. If arithmetic-expression-1 does not evaluate to an integer, the result is rounded up to the next whole number.

2) If arithmetic-expression-1 evaluates to 0 or a negative value, the data item referenced by data-name-2 is set to the predefined address NULL.

3) If data-name-1 is specified, the amount of storage to be allocated is the number of bytes required to hold an item as described by data-name-1. If a data description entry subordinate to data-name-1 contains an OCCURS DEPENDING ON clause, the maximum length of the record is allocated.

4) If the specified amount of storage is available for allocation, it shall be obtained and:

   a) if the RETURNING phrase is specified, the data item referenced by data-name-2 is set to the address of that storage,

   b) if data-name-1 is specified, the address of the based data item referenced by data-name-1 is set to the address of that storage.

5) If the specified amount of storage is not available for allocation:

   a) if the RETURNING phrase is specified, the data item referenced by data-name-2 is set to the predefined address NULL,

b) if data-name-1 is specified, the address of the based data item referenced by data-name-1 is set to the predefined address NULL,

c) the EC-STORAGE-NOT-AVAIL exception condition is set to exist.

6) If both the INITIALIZED phrase and arithmetic-expression-1 are specified, all bytes of the allocated storage are initialized to binary zeros.

7) If both the INITIALIZED phrase and data-name-1 are specified, the allocated storage is initialized as if an INITIALIZE data-name-1 WITH FILLER ALL TO VALUE THEN TO DEFAULT statement were executed.

8) If the INITIALIZED phrase is not specified and arithmetic-expression-1 is specified, the content of the allocated storage is undefined.

9) If the INITIALIZED phrase is not specified and data-name-1 is specified, data items of class object or class pointer in the allocated storage are initialized to null and the content of the other data items in the allocated storage is undefined.

10) The allocated storage persists until explicitly released with a FREE statement or the run unit is terminated, whichever occurs first.

14.9.4 CALL statement

The CALL statement causes control to be transferred to a specific program within the run unit.

14.9.4.1 General format

Format 1 (on-overflow):

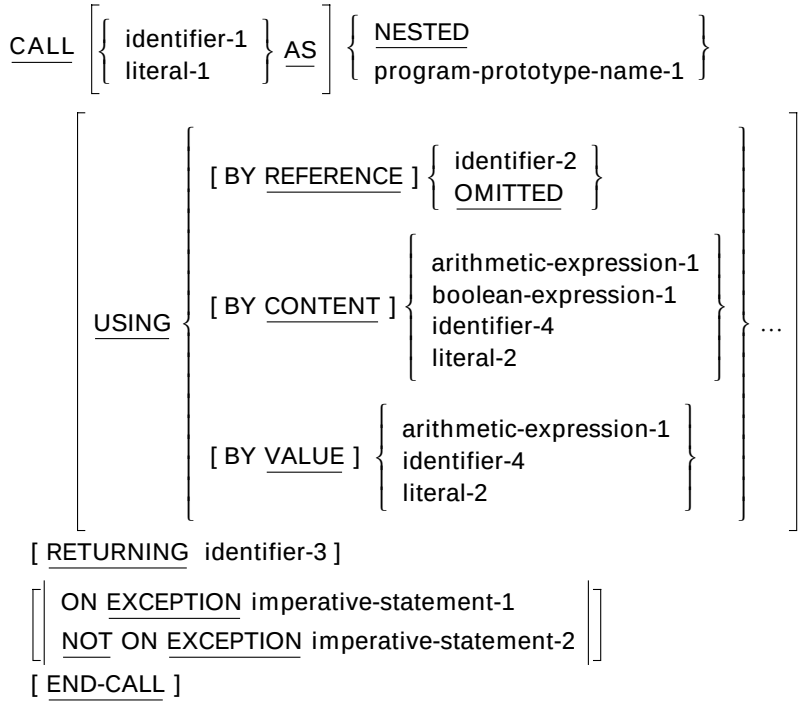
```
CALL { identifier-1 } [ USING { [ BY REFERENCE ] { identifier-2 } ... } ... ]
     { literal-1    }         { BY CONTENT { identifier-2 } ...        }
  [ RETURNING identifier-3 ]
  [ ON OVERFLOW imperative-statement-1 ]
  [ END-CALL ]
```

NOTE ON OVERFLOW is an archaic feature and its use should be avoided.

Format 2 (on-exception):

```
CALL { identifier-1 } [ USING { [ BY REFERENCE ] { identifier-2 } ... } ... ]
     { literal-1    }         { BY CONTENT { identifier-2 } ...        }
  [ RETURNING identifier-3 ]
  [ ON EXCEPTION imperative-statement-1 ]
  [ NOT ON EXCEPTION imperative-statement-2 ]
  [ END-CALL ]
```

Format 3 (program-prototype):



14.9.4.2 Syntax rules

ALL FORMATS

1) Identifier-1 shall be defined as an alphanumeric, national, or program-pointer data item.

2) Literal-1 shall be an alphanumeric or national literal and shall not be a zero-length literal.

3) Identifier-2 shall reference an address-identifier or a data item defined in the file, working-storage, local-storage, or linkage section. If the BY REFERENCE phrase is specified or implied, identifier-2 shall not be defined in the working-storage or file section of a factory or an instance object.

4) If the BY REFERENCE phrase is not specified or implied for an identifier-2 or if identifier-2 is an address-identifier, identifier-2 is a sending operand.

5) If the BY REFERENCE phrase is specified or implied for an identifier-2 and identifier-2 is not an address-identifier, it is a receiving operand.

6) If the BY REFERENCE phrase is specified or implied for an identifier-2 that is a bit data item, identifier-2 shall be described such that:

   a) subscripting and reference modification in identifier-2 consist of only fixed-point numeric literals or arithmetic expressions in which all operands are fixed-point numeric literals and the exponentiation operator is not specified; and

   b) it is aligned on a byte boundary.

7) Identifier-3 shall reference a data item defined in the file, working-storage, local-storage, or linkage section.

8) If identifier-3 is a bit data item, identifier-3 shall be described such that:

a) subscripting and reference modification in identifier-3 consist of only fixed-point numeric literals or arithmetic expressions in which all operands are fixed-point numeric literals and the exponentiation operator is not specified; and

b) it is aligned on a byte boundary.

9) Identifier-3 is a receiving operand.

FORMATS 1 AND 2

10) If the BY REFERENCE phrase is specified or implied for an identifier-2, that identifier shall be neither a strongly-typed group item nor a data item of class object or pointer.

11) Identifier-2 and identifier-3 shall not be described with the ANY LENGTH clause.

12) Identifier-2 shall not reference a variable-length group.

FORMAT 3

13) The NESTED phrase may be specified only in a program definition.

14) If identifier-1 references a restricted program-pointer, the signature of the program-prototype specified in the definition of that pointer shall be the same as the signature of program-prototype-name-1.

15) If the NESTED phrase is specified, literal-1 shall be specified. Literal-1 shall be the same as the program-name specified in a PROGRAM-ID paragraph of a common program as specified in 8.4.5.2, Scope of program-names, or of a program that is directly contained in the calling program.

16) Program-prototype-name-1 shall be specified in a program-specifier in the REPOSITORY paragraph.

17) Identifier-4 and any identifier specified in arithmetic-expression-1 or boolean-expression-1 is a sending operand.

18) Identifier-4 shall not be described with the ANY LENGTH clause.

19) If the BY CONTENT or BY REFERENCE phrase is specified or implied for an argument, the BY REFERENCE phrase shall be specified for the corresponding formal parameter in the procedure division header.

20) BY CONTENT shall not be omitted when identifier-4 is an identifier that is permitted as a receiving operand, except that BY CONTENT may be omitted when identifier-4 is an object property.

21) If the BY VALUE phrase is specified for an argument, the BY VALUE phrase shall be specified for the corresponding formal parameter in the procedure division header.

22) If identifier-4 or its corresponding formal parameter is specified with a BY VALUE phrase, identifier-4 shall be of class numeric, object, or pointer.

23) If literal-2 or its corresponding formal parameter is specified with the BY VALUE phrase, literal-2 shall be a numeric literal.

24) If the OMITTED phrase is specified, the OPTIONAL phrase shall be specified for the corresponding formal parameter in the procedure division header.

25) The rules for conformance specified in 14.8, Conformance for parameters and returning items, apply.

14.9.4.3 General rules

ALL FORMATS

1) The instance of the program, function, or method that executes the CALL statement is the activating runtime element.

2) The sequence of arguments in the USING phrase of the CALL statement and the sequence of formal parameters in the USING phrase of the called program's procedure division header determine the correspondence between arguments and formal parameters. This correspondence is positional and not by name equivalence.

   NOTE The first argument corresponds to the first formal parameter, the second to the second, and the nth to the nth.

   The effect of the USING phrase on the activated runtime element is described in 14.2, Procedure division structure, general rules.

3) Execution of the CALL statement proceeds as follows:

   a) Arithmetic-expression-1, boolean-expression-1, identifier-1, identifier-2, and identifier-4 are evaluated and item identification is done for identifier-3 at the beginning of the execution of the CALL statement. If an exception condition exists, no program is called and execution proceeds as specified in general rule 3g. If an exception condition does not exist, the values of identifier-2, identifier-4, arithmetic-expression-1, boolean-expression-1, or literal-2 are made available to the called program at the time control is transferred to that program.

   b) The program being called is identified by its program-name or its location, which are determined as follows:

      — If identifier-1 references an alphanumeric or national data item or literal-1 is specified, the value of literal-1 or the content of the data item referenced by identifier-1 is the program-name of the program being called, as described in 8.3.1.1.1, User-defined words.

      — If identifier-1 references a program-pointer data item, the data item referenced by identifier-1 contains the location of the program being called.

      — If neither identifier-1 nor literal-1 is specified, program-prototype-name-1 determines the externalized program-name of the program being called, according to the rules specified in 12.3.7, REPOSITORY paragraph.

      If the program being called is a COBOL program, the runtime system attempts to locate the program being called. When the program-name is used for locating the program, the rules specified in 8.4.5, Scope of names and 8.4.5.2, Scope of program-names, apply. If the program being called is not a COBOL program, the rules for program-name formation and for locating the program are defined by the implementor.

      If the data item referenced by identifier-1 contains the predefined address NULL, the EC-PROGRAM-PTR-NULL exception condition is set to exist. If the program cannot be located or identifier-1 references a zero-length item, the EC-PROGRAM-NOT-FOUND exception condition is set to exist If either the EC-PROGRAM-NOT-FOUND or the EC-PROGRAM-PTR-NULL exception condition exists, the program is not called, and execution continues as specified in general rule 3g.

   c) If the program is located but the resources necessary to execute the program are not available, the EC-PROGRAM-RESOURCES exception condition is set to exist, the program is not called, and execution continues as specified in general rule 3g. The runtime resources that are checked in order to determine the availability of the called program for execution are defined by the implementor.

   d) If the resources are available and the program being called is a COBOL program, the rules for conformance specified in 14.8, Conformance for parameters and returning items, apply. If a violation of these rules is detected, the EC-PROGRAM-ARG-MISMATCH exception condition is set to exist if checking for it is enabled in both the called program and calling runtime element, the program is not called, and execution continues as specified in general rule 3g.

e) If the program being called is in the active state and that program does not have the recursive attribute, the EC-PROGRAM-RECURSIVE-CALL exception condition is set to exist, the program is not called, and execution continues as specified in general rule 3g.

f) If a fatal exception condition has not been raised, the program specified by the CALL statement is made available for execution and control is transferred to the called program. If identifier-1 is defined as a program-pointer data item and contains an invalid program address, execution of the CALL statement is undefined. If the called program is a COBOL program, its execution is described in 14.2, Procedure division structure; otherwise the execution is defined by the implementor.

g) If the program was not successfully called and an exception condition was set to exist, one of the following actions occurs:

1. If the exception condition is any of the EC-PROGRAM exception conditions and an ON OVERFLOW or ON EXCEPTION phrase is specified in the CALL statement, control is transferred to imperative-statement-1. Execution then continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the CALL statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

2. If checking for the exception condition is enabled, and if the exception condition is one of the EC-PROGRAM exception conditions and neither an ON OVERFLOW nor an ON EXCEPTION phrase is specified or if the exception condition is not one of the EC-PROGRAM exception conditions, any declarative that is associated with that exception condition is executed. Execution then proceeds as defined for the exception condition and execution of the declarative. If control is returned from the declarative, the ON OVERFLOW, ON EXCEPTION, and NOT ON EXCEPTION phrases, if specified, are ignored. All other effects of the CALL statement are defined by the implementor.

3. If checking for the exception condition is not enabled, subsequent behavior is as specified in 14.6.12.1, Exception conditions.

h) If the program was successfully called, after control is returned from the called program the ON OVERFLOW or ON EXCEPTION phrase, if specified, is ignored. If an exception condition is propagated from the called program, execution continues as specified in 14.6.12.1, Exception conditions; otherwise, control is transferred to the end of the CALL statement or, if the NOT ON EXCEPTION phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative- statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the CALL statement.

4) If a RETURNING phrase is specified, the result of the activated program is placed into identifier-3.

FORMATS 1 AND 2

5) Both the BY CONTENT and BY REFERENCE phrases are transitive across the parameters that follow them until another BY CONTENT or BY REFERENCE phrase is encountered. If neither the BY CONTENT nor the BY REFERENCE phrase is specified prior to the first parameter, the BY REFERENCE phrase is assumed.

FORMAT 3

6) If the NESTED phrase is specified, the common or contained program that has literal-1 specified in the PROGRAM-ID paragraph is used to determine the characteristics of the called program.

7) If the NESTED phrase is not specified, program-prototype-name-1 is used to determine the characteristics of the called program.

8) An argument that consists merely of a single identifier or literal is regarded as an identifier or literal rather than an arithmetic or boolean expression.

9) If an argument is specified without any of the keywords BY REFERENCE, BY CONTENT, or BY VALUE, the manner used for passing this argument is determined as follows:

    a) When the BY REFERENCE phrase is specified or implied for the corresponding formal parameter:

        1. if the argument meets the requirements of syntax rule 3, BY REFERENCE is assumed;

        2. if the argument does not meet the requirements of syntax rule 3, BY CONTENT is assumed.

    b) When the BY VALUE phrase is specified or implied for the corresponding formal parameter, BY VALUE is assumed.

10) Control is transferred to the called program in a manner consistent with the entry convention specified for the program.

11) If an OMITTED phrase is specified or a trailing argument is omitted, the omitted-argument condition for that parameter evaluates to true in the called program. (8.8.4.1.7, Omitted-argument condition.)

12) If a parameter for which the omitted-argument condition is true is referenced in a called program, except as an argument or in the omitted-argument condition, the EC-PROGRAM-ARG-OMITTED exception condition is set to exist.

14.9.5 CANCEL statement

The CANCEL statement ensures that the next time the referenced program is called it will be in its initial state.

14.9.5.1 General format

```
CANCEL { identifier-1              } ...
       { literal-1                 }
       { program-prototype-name-1  }
```

14.9.5.2 Syntax rules

1) Identifier-1 shall be defined as an alphanumeric or national data item.

2) Literal-1 shall be an alphanumeric or national literal and shall not be a zero-length literal.

3) Program-prototype-name-1 shall be a program prototype specified in the REPOSITORY paragraph.

14.9.5.3 General rules

1) The program to be canceled is identified by one of the following:

   a) the content of the data item referenced by identifier-1,

   b) the value of literal-1,

   c) program-prototype-name-1.

   If identifier-1 or literal-1 is specified, 8.3.1.1.1, User-defined words, describes how this value is used to identify the program to be canceled.

2) The program-name is used by the runtime system to locate the program according to the rules specified in 8.4.5, Scope of names, and 8.4.5.2, Scope of program-names.

3) Subsequent to the execution of a CANCEL statement, the program referred to therein ceases to have any logical relationship to the run unit in which the CANCEL statement appears. If the program referenced by a successfully executed CANCEL statement in a run unit is subsequently called in that run unit, that program is in its initial state. (See 14.6.2, State of a function, method, object, or program.)

   NOTE It is neither prohibited nor required that the storage of the specified program be freed by the execution of a CANCEL statement.

4) When a CANCEL statement is executed, all programs contained within the program referenced by the CANCEL statement are also canceled. The result is the same as if an explicit CANCEL statement were executed for each contained program in the reverse order in which the programs appear in the outermost program.

5) The program to be canceled shall not be in the active state or contain a program in the active state. If a program in the active state is explicitly or implicitly referenced in a CANCEL statement and checking for EC-PROGRAM-CANCEL-ACTIVE is enabled in both the program to be canceled and the runtime element containing the CANCEL statement, the EC-PROGRAM-CANCEL-ACTIVE exception condition is raised in the runtime element containing the CANCEL statement and the referenced program is not canceled. If checking for EC-PROGRAM-CANCEL-ACTIVE is not enabled, the results of such a reference are defined by the implementor.

6) A logical relationship to a canceled program is established only by execution of a subsequent CALL statement referencing that program.

7) No action is taken when a CANCEL statement is executed referencing a program that has not been called in this run unit or has been called and is at present canceled. Control is transferred to the next executable statement following the explicit CANCEL statement.

8) The contents of data items in external data records described by a program are not changed when that program is canceled.

9) During execution of a CANCEL statement, an implicit CLOSE statement without any optional phrases is executed for each file-name associated with an internal file connector that is open in the program referenced in the CANCEL statement. These implicit CLOSE statements are executed for all such files, even when an error occurs during the execution of such CLOSE statements. Any USE procedures associated with any of these files are not executed.

10) If the program to be canceled is other than a COBOL program, the effects of the CANCEL statement are implementor-defined.

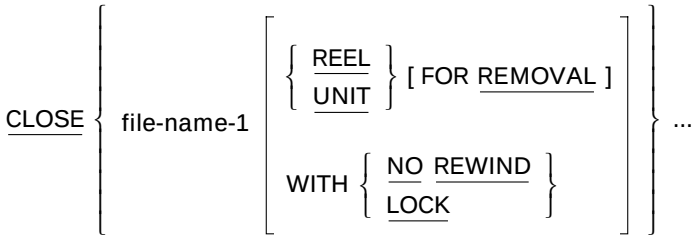
11) If a program-pointer has been set to point to the program to be canceled, the result of referencing the program-pointer in a subsequent CALL statement is undefined.

12) If identifier-1 references a zero-length item, the CANCEL statement has no effect.

14.9.6 CLOSE statement

The CLOSE statement terminates the processing of reels/units and files with rewind, lock, or removal where applicable.

14.9.6.1 General format



14.9.6.2 Syntax rules

1) The NO REWIND, REEL, and UNIT phrases may be used only with files that are of sequential organization.

2) The words REEL and UNIT are equivalent.

14.9.6.3 General rules

1) The file connector referenced by file-name-1 shall be open. If the file connector is not open, the CLOSE statement is unsuccessful and the I-O status indicator for the file connector is set to '42'.

2) For the purpose of showing the effect of various types of CLOSE statements as applied to various storage media, all files are divided into the following categories, where the term 'file' means the physical file:

   a) Non-unit. A file whose input or output medium is such that the concepts of rewind and units have no meaning.

   b) Sequential single-unit. A sequential file that is entirely contained on one unit.

   c) Sequential multi-unit. A sequential file that is contained on more than one unit.

   d) Non-sequential single/multi-unit. A file with organization other than sequential, that resides on a mass storage device.

3) The results of executing each type of CLOSE for each category of physical file are summarized in table 15, Relationship of categories of physical files and the format of the CLOSE statement.

Table 15 — Relationship of categories of physical files and the format of the CLOSE statement

CLOSE statement format	File category			
	Non-unit	Sequential single-unit	Sequential multi-unit	Non-sequential single/multi-unit
CLOSE	c	c,g	a,c,g	c
CLOSE WITH LOCK	c,e	c,e,g	a,c,e,g	c,e
CLOSE WITH NO REWIND	c,h	b,c	a,b,c	N/A
CLOSE UNIT	f	f,g	f,g	N/A
CLOSE UNIT FOR REMOVAL	f	d,f,g	d,f,g	N/A

The definitions of the symbols in table 15, Relationship of categories of physical files and the format of the CLOSE statement, are given below. The notation 'N/A' means that the combination is not applicable. The other symbols apply to the rules below. Where the definition depends on whether the file is an input, output, or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

a) Effect on previous units

Input files and input-output files:

All units in the physical file prior to the current unit are closed except those units controlled by a prior CLOSE UNIT statement. If the current unit is not the last in the physical file, the units in the physical file following the current one are not processed.

Output files:

All units in the physical file prior to the current unit are closed except those units controlled by a prior CLOSE UNIT statement.

b) No rewind of current reel

The current unit is left in its current position.

c) Close file

Closing operations specified by the implementor are executed.

d) Unit removal

The current unit is rewound, when applicable, and the unit is logically removed from the run unit; however, the unit may be accessed again, in its proper order of units within the physical file, if a CLOSE statement without the UNIT phrase is subsequently executed for this file followed by the execution of an OPEN statement for the file.

NOTE This final committee draft International Standard does not address when the unit is unloaded or left loaded.

e) Close lock

When the LOCK phrase is specified, the file connector referenced by file-name-1 shall not be opened again during the execution of this run unit.

f) Close unit

Input files and input-output files (unit media):

1. If the current unit is the last or only unit for the physical file, there is no unit swap, the current volume pointer remains unchanged, and the file position indicator is set to indicate that no next or previous logical record exists.

2. If another unit exists for the physical file, a unit swap occurs, the current volume pointer is updated to point to the next unit existing in the physical file, and the file position indicator is set to one less than the number of the first record existing on the new current volume. If no records exist for the current volume, another unit swap occurs.

Output files (unit media):

A unit swap occurs and the current volume pointer is updated to point to the new unit.

Input files, input-output files, and output files (non-unit media):

Execution of this statement is considered successful. The file remains in the open mode, the file position indicator is unchanged, the I-O status indicator for the file connector is set to '07', and other no action takes place.

g) Rewind

The current reel or analogous device is positioned at its physical beginning.

h) Optional phrases ignored

The CLOSE statement is executed as if none of the optional phrases were present. The I-O status indicator for the file connector is set to '07'.

4) The execution of the CLOSE statement causes the value of the I-O status associated with file-name-1 to be updated as specified in 9.1.12, I-O status.

5) No report associated with a report file that is referenced in the CLOSE statement shall be in the active state. If any report is in the active state, the CLOSE statement for that file is completed and the EC-REPORT-NOT-TERMINATED exception condition is set to exist.

6) If the file position indicator of the file connector referenced by file-name-1 is set to indicate that an optional input file is not present, no end-of-file or unit processing is performed for the file and the file position indicator and the current volume pointer are unchanged.

7) The availability of the record area associated with file-name-1 to the runtime element depends on the successful or unsuccessful execution of the CLOSE statement without the UNIT phrase and whether file-name-1 is referenced in a SAME RECORD AREA clause. If file-name-1 is specified in a SAME RECORD AREA clause, the record area is available to the runtime element if any of the file connectors referenced by the other file-names in that SAME RECORD AREA clause are open. If none of these file connectors is open or if file-name-1 is not specified in a SAME RECORD AREA clause, the successful execution of a CLOSE statement makes the record area unavailable to the runtime element and the unsuccessful execution of the CLOSE statement makes the availability of the record area undefined.

8) Following the successful execution of a CLOSE statement without the UNIT phrase, the physical file is no longer associated with the file connector referenced by file-name-1 and the open mode of that file connector is set such that it is no longer in an open mode.

9) The file lock and any record locks associated with the file connector referenced by file-name-1 are released by the execution of the CLOSE statement.

10) If more than one file-name-1 is specified in a CLOSE statement, the result of executing this CLOSE statement is the same as if a separate CLOSE statement had been written for each file-name-1 in the same order as specified in the CLOSE statement. If an implicit CLOSE statement results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, processing resumes at the next implicit CLOSE statement, if any.

14.9.7 COMPUTE statement

The COMPUTE statement assigns to one or more data items the value of an arithmetic or boolean expression.

14.9.7.1 General format

Format 1 (arithmetic-compute):

```
COMPUTE { identifier-1 [ rounded-phrase ] } ... = arithmetic-expression-1
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-COMPUTE ]
```

Format 2 (boolean-compute):

```
COMPUTE { identifier-2 } ... = boolean-expression-1 [ END-COMPUTE ]
```

where rounded-phrase is described in 14.7.3, ROUNDED phrase.

14.9.7.2 Syntax rules

FORMAT 1

1) Identifier-1 shall reference either an elementary numeric item or an elementary numeric-edited item.

FORMAT 2

2) Identifier-2 shall reference an elementary boolean data item.

3) Boolean-expression-1 shall not consist solely of the figurative constant ALL literal.

14.9.7.3 General rules

1) The execution of a COMPUTE statement consists of the determination of a value and the subsequent storing of that value. This proceeds as follows:

   a) If arithmetic-expression-1 consists of a single fixed-point numeric literal or a single fixed-point numeric data item, arithmetic-expression-1 evaluates to the exact value of that literal or identifier, before the application of any rounding, truncation, or decimal point alignment applicable for the COMPUTE statement and mode of arithmetic in effect. The resulting value is then stored in each data item referenced by identifier-1.

   b) If arithmentic-expression-1 consists of anything other than a single fixed-point numeric literal or a single fixed-point numeric data item, arithmetic-expression-1 is evaluated to produce the resulting value. The resulting value is then stored in each data item referenced by identifier-1.

   c) Boolean-expression-1 evaluates to the value of that boolean expression. The number of boolean positions in the value resulting from the evaluation of boolean-expression-1 is the number of boolean positions in the largest boolean item referenced in the expression. The resulting value is then stored in each data item referenced by identifier-2.

2) If a receiving operand is defined by the same data description entry as a sending operand, general rule 1 defines the result of the operation.

3) Additional rules and explanations relative to this statement are given in 8.8.1, Arithmetic expressions; 8.8.2, Boolean expressions; 14.6.12.2, Incompatible data; 14.7.3, ROUNDED phrase; 14.7.4, SIZE ERROR phrase and size error condition; and 14.7.6, Arithmetic statements.

14.9.8 CONTINUE statement

The CONTINUE statement is a no-operation statement. It indicates that no executable statement is present.

14.9.8.1 General format

CONTINUE

14.9.8.2 Syntax rules

1) The CONTINUE statement may be used anywhere a conditional statement or an imperative-statement may be used.

14.9.8.3 General rules

1) The CONTINUE statement has no effect on the execution of the runtime element.

14.9.9 DELETE statement

The DELETE statement logically removes a record from a mass storage file.

14.9.9.1 General format

```
DELETE file-name-1  RECORD
  [ retry-phrase ]
  [ | INVALID KEY imperative-statement-1                | ]
  [ | NOT INVALID KEY imperative-statement-2            | ]
  [ END-DELETE ]
```

where retry-phrase is described in 14.7.8, RETRY phrase

14.9.9.2 Syntax rules

1) The DELETE statement shall not be specified for a file with sequential organization.

2) The INVALID KEY and the NOT INVALID KEY phrases shall not be specified for a DELETE statement that references a file that is in sequential access mode.

14.9.9.3 General rules

1) The open mode of the file connector referenced by file-name-1 shall be I-O and the physical file associated with that file connector shall be a mass storage file.

2) For a file that is in the sequential access mode, the last input-output statement executed for file-name-1 prior to the execution of the DELETE statement shall have been a successfully executed READ statement. The mass storage control system (MSCS) logically removes from the physical file the record that was accessed by that READ statement.

3) If the file is indexed and the access mode is random or dynamic, the mass storage control system (MSCS) logically removes from the physical file the record identified by the content of the prime record key data item associated with file-name-1. If the physical file does not contain the record specified by the key, the invalid key condition exists. (See 9.1.13, Invalid key condition.)

4) If the file is relative and the access mode is random or dynamic, the mass storage control system (MSCS) logically removes from the physical file that record identified by the content of the relative key data item associated with file-name-1. If the physical file does not contain the record specified by the key, the invalid key condition exists. (See 9.1.13, Invalid key condition.)

5) After the successful execution of a DELETE statement, the identified record has been logically removed from the physical file and can no longer be accessed.

6) If record locking is enabled for the file connector referenced by file-name-1 and the record identified for deletion is locked by another file connector, the result of the operation depends on the presence or absence of the RETRY phrase. If the RETRY phrase is specified, additional attempts may be made to delete the record as specified in the rules in 14.7.8, RETRY phrase. If the RETRY phrase is not specified or the record is not successfully removed as specified by the RETRY phrase, the record operation conflict condition exists. The I-O status is set in accordance with the rules for the RETRY phrase.

   When the record operation conflict condition exists as a result of the DELETE statement:

   a) The record is not logically removed, and may be accessed.

b) A value is placed into the I-O status associated with file-name-1 to indicate the record operation conflict condition.

c) The DELETE statement is unsuccessful.

7) If record locks are in effect, the following actions take place:

a) If single record locking is specified for the file connector associated with filename-1:

1. A lock held by that file connector on the deleted record is released at the completion of the successful execution of the DELETE statement.

2. A lock held by that file connector on another record is released at the beginning of the execution of the DELETE statement.

b) If multiple record locking is specified for the file connector associated with file-name-1, all locks held on the deleted record are released at the completion of the successful execution of the DELETE statement.

8) The execution of a DELETE statement does not affect the content of the record area or the content of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with file-name-1.

9) The file position indicator is not affected by the execution of a DELETE statement.

10) The execution of the DELETE statement causes the value of the I-O status associated with file-name-1 to be updated as specified in 9.1.12, I-O status.

11) Transfer of control following the successful or unsuccessful execution of the DELETE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the DELETE statement as specified in 9.1.13, Invalid key condition.

14.9.10 DISPLAY statement

The device format of the DISPLAY statement causes data to be transferred to a hardware or software device in the operating environment.

The screen format of the DISPLAY statement causes data associated with a literal or data item that is referenced in a screen item to be made available to the specified screen item and to be displayed on the terminal screen with specified attributes and at the specified position.

14.9.10.1 General format

Format 1 (device):

```
DISPLAY { identifier-1 } ... [ UPON mnemonic-name-1 ] [ WITH NO ADVANCING ] [ END-DISPLAY ]
        { literal-1    }
```

Format 2 (screen):

```
DISPLAY screen-name-1
[ AT { LINE NUMBER { identifier-2 }                                          } ]
     {             { integer-1   }                                           }
     { { COLUMN }                                                           }
     { { COL    } NUMBER { identifier-3 }                                   }
     {                   { integer-2   }                                    }
[ { ON EXCEPTION imperative-statement-1     } ]
  { NOT ON EXCEPTION imperative-statement-2 }
[ END-DISPLAY ]
```

14.9.10.2 Syntax rules

FORMAT 1

1) Identifier-1 shall not reference a data item of class object or class pointer.

2) Mnemonic-name-1 shall be specified in the SPECIAL-NAMES paragraph of the environment division and shall be associated with an implementor-defined device-name that is identified in the operating environment as a hardware or software device capable of receiving data from the program.

FORMAT 2

3) Identifier-2 and identifier-3 shall be unsigned integer data items.

14.9.10.3 General rules

FORMAT 1

1) The DISPLAY statement causes the content of each operand to be transferred to the device in the order listed. If an operand is a zero-length data item or a zero-length literal, no data is transferred for that operand. Any conversion of data required between literal-1 or the data item referenced by identifier-1 and the device is defined by the implementor.

2) The implementor shall define, for each device, the size of a data transfer.

3) If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed. A figurative constant other than ALL national literal shall be the alphanumeric representation of that figurative constant.

4) If the device is capable of receiving data of the same size as the data item being transferred, then the data item is transferred.

5) If a device is not capable of receiving data of the same size as the data item being transferred, then one of the following applies:

   a) If the size of the data item being transferred exceeds the size of the data that the device is capable of receiving in a single transfer, the data beginning with the leftmost character is stored aligned to the left in the receiving device, and the remaining data is then transferred according to general rules 4 and 5 until all the data has been transferred.

   b) If the size of the data item that the device is capable of receiving exceeds the size of the data being transferred, the transferred data is stored aligned to the left in the receiving device.

6) When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered without modifying the positioning of the device between the successive operands.

7) If identifier-1 references a variable-length group, the format in which its contents are displayed is defined by the implementor.

8) If the UPON phrase is not specified, the implementor's standard display device is used.

9) If the WITH NO ADVANCING phrase is specified, then the positioning of the device shall not be reset to the next line or changed in any other way following the display of the last operand. If the device is capable of positioning to a specific character position, it will remain positioned at the character position immediately following the last character of the last operand displayed. If the device is not capable of positioning to a specific character position, only the vertical position, if applicable, is affected. This may cause overprinting if the device supports overprinting.

10) If the WITH NO ADVANCING phrase is not specified, then after the last operand has been transferred to the device, the positioning of the device shall be reset to the leftmost position of the next line of the device.

11) If vertical positioning is not applicable on the device, the vertical positioning shall be ignored.

FORMAT 2

12) Column and line number positions are specified in terms of alphanumeric character positions.

13) The DISPLAY statement causes the transfer of data in accordance with the MOVE statement rules to each elementary screen item that is subordinate to screen-name-1 and is specified with the FROM, USING, or VALUE clause, from the data item or literal referenced in the FROM, USING, or VALUE clause. For the purpose of these specifications, all such screen items are considered to be referenced by the DISPLAY screen statement. If two or more of these elementary screen items overlap, the EC-SCREEN-FIELD-OVERLAP exception condition is set to exist. The transfer of data to the elementary screen items is done in the order that the screen items are specified within screen-name-1.

   NOTE When two screen items overlap, the display on the screen for the common character positions is determined by the second screen item specified within screen-name-1.

The transfer takes place and each elementary screen item is displayed on the terminal display subject to any editing implied in the character-string specified in the PICTURE clause of each elementary screen description entry.

14) The LINE and COLUMN phrases give the position on the terminal display screen at which the screen record associated with screen-name-1 is to start. The position is relative to the leftmost character column in the topmost line of the display that is identified as column 1 of line 1. Each subordinate elementary screen item is located relative to the start of the containing screen record. Identifier-2 and identifier-3 are evaluated once at the start of execution of the statement.

15) If the LINE phrase is not specified, the screen record starts on line 1.

16) If the COLUMN phrase is not specified, the screen record starts in column 1.

17) If the execution of the DISPLAY statement is successful, the ON EXCEPTION phrase, if specified, is ignored and control is transferred to the end of the DISPLAY statement or, if the NOT ON EXCEPTION phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the DISPLAY statement.

18) If the execution of the DISPLAY statement is unsuccessful, then:

    a) If the ON EXCEPTION phrase is specified in the DISPLAY statement, control is transferred to imperative-statement-1. Execution then continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the DISPLAY statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

    b) If the ON EXCEPTION phrase is not specified in the DISPLAY statement and an applicable declarative exists, control is transferred to that declarative and, if control is returned from that declarative control is transferred to the end of the DISPLAY statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

    c) If the ON EXCEPTION phrase is not specified in the DISPLAY statement and there is no applicable declarative, control is transferred to the end of the DISPLAY statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

19) If one or more of the exception conditions EC-SCREEN-FIELD-OVERLAP, EC-SCREEN-ITEM-TRUNCATED, EC-SCREEN-LINE-NUMBER, or EC-SCREEN-STARTING-COLUMN exists during the execution of the DISPLAY statement, the execution of the DISPLAY statement continues as described for that exception condition and upon completion, the execution of the DISPLAY statement is considered unsuccessful.

14.9.11 DIVIDE statement

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder.

14.9.11.1 General format

Format 1 (into):

```
DIVIDE { identifier-1 } INTO { identifier-2 [ rounded-phrase ] } ...
       { literal-1    }
  [ ON SIZE ERROR imperative-statement-1     ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-DIVIDE ]
```

Format 2 (into-giving):

```
DIVIDE { identifier-1 } INTO { identifier-2 }
       { literal-1    }     { literal-2    }
  GIVING { identifier-3 [ rounded-phrase ] } ...
  [ ON SIZE ERROR imperative-statement-1     ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-DIVIDE ]
```

Format 3 (by-giving):

```
DIVIDE { identifier-2 } BY { identifier-1 }
       { literal-2    }    { literal-1    }
  GIVING { identifier-3 [ rounded-phrase ] } ...
  [ ON SIZE ERROR imperative-statement-1     ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-DIVIDE ]
```

Format 4 (into-remainder):

```
DIVIDE { identifier-1 } INTO { identifier-2 }
       { literal-1    }     { literal-2    }
  GIVING identifier-3 [ rounded-phrase ]
  REMAINDER identifier-4
  [ ON SIZE ERROR imperative-statement-1     ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-DIVIDE ]
```

Format 5 (by-remainder):

```
DIVIDE { identifier-2 } BY { identifier-1 }
       { literal-2    }    { literal-1    }
  GIVING identifier-3 [ rounded-phrase ]
  REMAINDER identifier-4
  [ ON SIZE ERROR imperative-statement-1 ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
[ END-DIVIDE ]
```

where rounded-phrase is described in 14.7.3, ROUNDED phrase.

14.9.11.2 Syntax rules

1) Identifier-1 and identifier-2 shall reference an elementary data item of category numeric.

2) Identifier-3 and identifier-4 shall reference an elementary data item of category numeric or numeric-edited.

3) Literal-1 and literal-2 shall be numeric literals.

4) When native arithmetic is in effect, the composite of operands described in 14.7.6, Arithmetic statements, is determined by using all of the operands in the statement excluding the data item that follows the word REMAINDER.

14.9.11.3 General rules

ALL FORMATS

1) When native arithmetic is in effect, the quotient is the result of dividing the dividend by the divisor. When standard arithmetic, standard-decimal arithmetic, or standard-binary arithmetic is in effect, the quotient, the quotient is the result of the arithmetic expression

   (dividend / divisor)

   where the values of dividend and divisor are defined in the following general rules.

2) The process of determining the dividend and determining the divisor consists of the following:

   a) The dividend is identifier-2 or literal-2. The divisor is identifier-1 or literal-1.

   b) If an identifier is specified, item identification is done and the content of the resulting data item is the dividend or divisor.

   c) If a literal is specified, the value of the literal is the dividend or divisor.

3) Additional rules and explanations relative to this statement are given in 14.6.12.2, Incompatible data; 14.7.3, ROUNDED phrase; 14.7.4, SIZE ERROR phrase and size error condition; and 14.7.6, Arithmetic statements.

FORMAT 1

4) The evaluation proceeds in the following order:

   a) The initial evaluation consists of determining the divisor.

b) This divisor is used with each dividend, which is each identifier-2 proceeding from left to right. Item identification for identifier-2 is done as each dividend is determined. The quotient is then formed as specified in general rule 1 and stored in the corresponding identifier-2 as indicated in 14.7.6, Arithmetic statements.

FORMATS 2 AND 3

5) The evaluation proceeds in the following order:

a) The initial evaluation is determining the divisor and determining the dividend.

b) The quotient is then formed as specified in general rule 1 and stored in each identifier-3 as specified in 14.7.6, Arithmetic statements.

FORMATS 4 AND 5

6) The evaluation proceeds in the following order:

a) The initial evaluation is determining the divisor and determining the dividend.

b) The quotient is then formed as specified in general rule 1 and stored in identifier-3 as specified in 14.7.6, Arithmetic statements.

c) If the size error condition is not raised, a subsidiary quotient is developed that is signed and derived from the quotient by truncation of digits at the least significant end and that has the same number of digits and the same decimal point location as the data item referenced by identifier-3. The remainder is calculated as indicated in general rule 7 and is stored in the data item referenced by identifier-4 unless storing the value would cause a size error condition, in which case the content of identifier-4 is unchanged and execution proceeds as indicated in 14.7.4, SIZE ERROR phrase and size error condition. Item identification of the data item referenced by identifier-4 is done after the quotient is stored in the data item referenced by identifier-3.

7) When native arithmetic is in effect, the remainder is the result of multiplying the subsidiary quotient and the divisor and subtracting the product from the dividend. When standard arithmetic, standard-decimal arithmetic, or standard-binary arithmetic is in effect, the remainder is the result of the arithmetic expression

(dividend - (subsidiary-quotient * divisor))

where the values of dividend and divisor are defined in general rule 2 and where subsidiary-quotient represents the subsidiary quotient as defined in general rule 6.

14.9.12 EVALUATE statement

The EVALUATE statement describes a multi-branch, multi-join structure. It may cause multiple conditions to be evaluated. The subsequent action of the runtime element depends on the results of these evaluations.

14.9.12.1 General format

```
EVALUATE selection-subject [ ALSO selection-subject ] ...
  { { WHEN selection-object [ ALSO selection-object ] ... } ... imperative-statement-1 } ...
  [ WHEN OTHER imperative-statement-2 ]
  [ END-EVALUATE ]
```

where selection-subject is:

```
{ identifier-1
  literal-1
  arithmetic-expression-1
  boolean-expression-1
  condition-1
  TRUE
  FALSE }
```

where selection-object is:

```
{ [ NOT ] identifier-2
  [ NOT ] literal-2
  [ NOT ] arithmetic-expression-2
  [ NOT ] boolean-expression-2
  [ NOT ] range-expression
  condition-2
  partial-expression-1
  TRUE
  FALSE
  ANY }
```

where range-expression is:

```
{ identifier-3 | literal-3 | arithmetic-expression-3 } { THROUGH | THRU } { identifier-4 | literal-4 | arithmetic-expression-4 } [ IN alphabet-name-1 ]
```

14.9.12.2 Syntax rules

1) The words THROUGH and THRU are equivalent.

2) The number of selection objects within each set of selection objects shall be equal to the number of selection subjects.

3) Alphabet-name-1 may be specified only when the literals or identifiers specified in the THROUGH phrase are of class alphabetic, alphanumeric, or national. If literal-3 or identifier-3 is of class national, alphabet-name-1 shall reference an alphabet that defines a national collating sequence; otherwise, alphabet-name-1 shall reference an alphabet that defines an alphanumeric collating sequence.

4) The two operands in a range-expression shall be of the same class and shall not be of class boolean, object or pointer.

5) A selection object is a partial-expression if the leftmost portion of the selection object is a relational operator, a class condition without the identifier, a sign condition without the identifier, or a sign condition without the arithmetic expression.

6) The classification of some selection subjects or selection objects is changed for a particular WHEN phrase as follows:

   a) If the selection subject is TRUE or FALSE and the selection object is a boolean expression that results in one boolean character, the selection object is treated as a boolean condition and therefore condition-2.

   b) If the selection object is TRUE or FALSE and the selection subject is a boolean expression that results in one boolean character, the selection subject is a treated as boolean condition and therefore condition-1.

   c) If the selection subject is other than TRUE or FALSE and the selection object is a boolean expression that results in one boolean character, the selection object is treated as a boolean expression and therefore boolean-expression-2.

   d) If the selection object is other than TRUE or FALSE and the selection subject is a boolean expression that results in one boolean character, the selection subject is treated as a boolean expression and therefore boolean-expression-1.

   e) If the selection object is a partial expression and the selection subject is a data item of the class boolean or numeric, the selection subject is treated as an identifier.

7) Each selection object within a set of selection objects shall correspond to the selection subject having the same ordinal position within the set of selection subjects according to the following rules:

   a) Identifiers, literals, or expressions appearing within a selection object shall be valid operands for comparison to the corresponding operand in the set of selection subjects in accordance with 8.8.4.1.1, Relation conditions.

   b) Condition-2 or the words TRUE or FALSE appearing as a selection object shall correspond to condition-1 or the words TRUE or FALSE in the set of selection subjects.

   c) The word ANY may correspond to a selection subject of any type.

   d) Partial-expression-1 shall correspond to a selection subject that is an identifier, a literal, an arithmetic expression, or a boolean expression. Partial-expression-1 shall be a sequence of COBOL words such that, were it preceded by the corresponding selection subject, a conditional expression would result.

8) If a selection object is specified by partial-expression-1, that selection object is treated as though it were specified as condition-2, where condition-2 is the conditional expression that results from preceding partial-expression-1 by the selection subject. The corresponding selection subject is treated as though it were specified by the word TRUE.

9) Neither identifier-3 nor identifier-4 shall reference a variable-length group.

10) The permissible combinations of selection subject and selection object operands are indicated in table 16, Combination of operands in the EVALUATE statement.

Table 16 — Combination of operands in the EVALUATE statement

Selection object	Selection subject					
	Identifier	**Literal**	**Arithmetic expression**	**Boolean expression**	**Condition**	**TRUE or FALSE**
[NOT] identifier	Y	Y	Y	Y		
[NOT] literal	Y		Y	Y		
[NOT] arithmetic-expression	Y	Y	Y			
[NOT] boolean-expression	Y	Y		Y		
[NOT] range-expression	Y	Y	Y			
Condition					Y	Y
Partial-expression	Y	Y	Y	Y		
TRUE or FALSE					Y	Y
ANY	Y	Y	Y	Y	Y	Y
The letter 'Y' indicates a permissible combination. A space indicates an invalid combination.						

14.9.12.3 General rules

1) If an operand of the EVALUATE statement consists of a single literal, that operand is treated as a literal, not as an expression.

2) The manner of determining the range of values specified by the THROUGH phrase is described in 14.7.7, THROUGH phrase.

3) At the beginning of the execution of the EVALUATE statement, each selection subject is evaluated and assigned a value, a range of values, or a truth value as follows:

   a) Any selection subject specified by identifier-1 is assigned the value and class of the data item referenced by the identifier. If the selection subject is a numeric data item or a boolean data item whose length is one boolean position, the selection subject for this evaluation is treated as identifier-1 and not an arithmetic or boolean expression.

   b) Any selection subject specified by literal-1 is assigned the value and class of the specified literal.

   c) Any selection subject specified by arithmetic-expression-1 is assigned a numeric value according to the rules for evaluating an arithmetic expression.

   d) Any selection subject in which boolean-expression-1 is specified is assigned a boolean value according to the rules for evaluating boolean expressions.

   e) Any selection subject specified by condition-1 is assigned a truth value according to the rules for evaluating conditional expressions.

   f) Any selection subject specified by the words TRUE or FALSE is assigned a truth value. The truth value 'true' is assigned to those items specified with the word TRUE, and the truth value 'false' is assigned to those items specified with the word FALSE.

4) The execution of the EVALUATE statement proceeds by processing each WHEN phrase from left to right in the following manner:

a) Each selection object within the set of selection objects for each WHEN phrase is paired with the selection subject having the same ordinal position within the set of selection subjects. The result of the analysis of this set of selection subjects and objects is either true or false as follows:

   1. If the selection object is the word ANY, the result is true.

   2. If the selection object is partial-expression-1, the selection subject is placed to the left of the leading relational operator and the resulting conditional expression is evaluated. The result of the evaluation is the truth value of the expression.

   3. If the selection object is condition-2, the selection subject is either TRUE or FALSE. If the truth value of the selection subject and selection object match, the result of the analysis is true. If they do not match, the result is false.

   4. If the selection object is either TRUE or FALSE, the selection subject is condition-1. If the truth value of the selection subject and selection object match, the result of the analysis is true. If they do not match, the result is false.

   5. If the selection object is a range-expression, the pair is considered to be a conditional expression of one of the following forms:

      when 'NOT' is not specified in the selection object;

         selection-subject >= left-part AND selection-subject <= right-part

      when 'NOT' is specified in the selection object

         selection-subject < left-part OR selection-subject > right-part

      where left-part is identifier-3, literal-3, or arithmetic-expression-3 and right-part is identifier-4, literal-4, or arithmetic-expression-4. The result of the analysis is the truth value of the resulting conditional expression.

   6. If the selection object is identifier-2, literal-2, arithmetic-expression-2, or boolean-expression-2, the pair is considered to be a conditional expression of the following form:

         selection-subject [NOT] = selection-object

      where 'NOT' is present if it is present in the selection object. The result of the analysis is the truth value of the resulting conditional expression.

b) If the result of the analysis is true for every pair in a WHEN phrase, that WHEN phrase satisfies the set of selection subjects and no more WHEN phrases are analyzed.

c) If the result of the analysis is false for any pair in a WHEN phrase, no more pairs in that WHEN phrase are evaluated and the WHEN phrase does not match the set of selection subjects.

d) This procedure is repeated for subsequent WHEN phrases, in the order of their appearance in the source element, until either a WHEN phrase satisfying the set of selection subjects is selected or until all sets of selection objects are exhausted.

5) The execution of the EVALUATE statement then proceeds as follows:

   a) If a WHEN phrase is selected, execution continues with the first imperative-statement-1 following the selected WHEN phrase.

   b) If no WHEN phrase is selected and a WHEN OTHER phrase is specified, execution continues with imperative-statement-2.

c) The execution of the EVALUATE statement is terminated when execution reaches the end of imperative-statement-1 of the selected WHEN phrase or the end of imperative-statement-2, or when no WHEN phrase is selected and no WHEN OTHER phrase is specified.

14.9.13 EXIT statement

The EXIT statement provides a common end point for a series of procedures.

The EXIT PROGRAM statement marks the logical end of a called program.

The EXIT METHOD statement marks the logical end of an invoked method.

The EXIT FUNCTION statement marks the logical end of the execution of a function.

The EXIT PERFORM statement provides a means of exiting an inline PERFORM (with or without returning to any specified test).

The EXIT PARAGRAPH and EXIT SECTION statements provide a means of exiting a structured procedure without executing any of the following statements within the procedure.

14.9.13.1 General format

Format 1 (simple):

```
EXIT
```

Format 2 (program):

```
EXIT PROGRAM [ RAISING { EXCEPTION exception-name-1 }
                       { identifier-1               }
                       { LAST EXCEPTION             } ]
```

Format 3 (method):

```
EXIT METHOD [ RAISING { EXCEPTION exception-name-1 } ]
                      { identifier-1               }
                      { LAST EXCEPTION             }
```

Format 4 (function):

```
EXIT FUNCTION [ RAISING { EXCEPTION exception-name-1 } ]
                        { identifier-1               }
                        { LAST EXCEPTION             } ]
```

Format 5 (inline-perform):

```
EXIT PERFORM [ CYCLE ]
```

Format 6 (procedure):

```
EXIT { PARAGRAPH }
     { SECTION   }
```

14.9.13.2 Syntax rules

FORMAT 1

1) The EXIT statement shall appear in a sentence by itself that shall be the only sentence in the paragraph.

FORMATS 2, 3, AND 4

2) The EXIT statement shall not be specified in a declarative procedure for which the GLOBAL phrase is specified in the associated USE statement.

3) Exception-name-1 shall be a level-3 exception-name as specified in the rules for exception conditions. (See 14.6.12.1, Exception conditions.)

   If exception-name-1 is a level-3 exception-name for EC-USER, exception-name-1 shall be specified in the RAISING phrase of the procedure division header of the source element in which this EXIT statement is contained.

4) Identifier-1 is a sending operand.

5) Identifier-1 shall be an object reference, subject to the following constraints:

   a) If the data description entry of identifier-1 specifies a class-name, the class identified by that class-name or one of the superclasses of that class is specified in the RAISING phrase of the procedure division header of the source element containing this EXIT statement and the presence or absence of the FACTORY phrase is the same in the data description entry of identifier-1 as in the RAISING phrase of the procedure division header of the source element that contains this EXIT statement.

   b) If the data description entry of identifier-1 specifies an interface-name, the interface referenced by that interface-name shall conform to an interface specified in the RAISING phrase of the procedure division header of the source element that contains this EXIT statement.

   c) If the data description entry of identifier-1 specifies an ACTIVE-CLASS phrase, the class of the object containing this EXIT statement and the presence or absence of the FACTORY phrase is the same in the data description entry of identifier-1 as in the RAISING phrase of the procedure division header of the source element that contains this EXIT statement.

   d) Identifier-1 shall not be a universal object reference.

6) The LAST phrase may be specified only in a declarative procedure.

FORMAT 2

7) An EXIT PROGRAM statement may be specified only in a program procedure division.

FORMAT 3

8) An EXIT METHOD statement may be specified only in a method procedure division.

FORMAT 4

9) An EXIT FUNCTION statement may be specified only in a function procedure division.

FORMAT 5

10) The EXIT PERFORM statement may be specified only in an inline PERFORM statement.

FORMAT 6

11) The EXIT statement with the SECTION phrase may be specified only in a section.

14.9.13.3 General rules

FORMAT 1

1) An EXIT statement serves only to enable the user to assign a procedure-name to a given point in a procedure division. Such an EXIT statement has no other effect on the compilation or execution.

FORMATS 2, 3, AND 4

2) If an EXIT statement is executed within the range of a declarative procedure whose USE statement contains the GLOBAL phrase and that USE statement is specified in the same function, method, or program as the function format, method format, or program format EXIT statement respectively, the EC-FLOW-GLOBAL-EXIT exception condition is set to exist.

3) If the RAISING phrase is specified, an exception condition is raised in the activating runtime element if checking for that exception condition is enabled in the activating runtime element, and execution continues in that runtime element as specified in the rules for the activating statement after the result, if any, of the activated element is returned to the activating element. The exception condition that exists is determined as follows:

   a) If exception-name-1 is specified, it is that exception condition.

   b) If identifier-1 is specified, the object referenced by identifier-1 becomes the current exception object in the activating runtime element. Additional rules are specified in 14.6.12.1.4, Exception objects.

   c) If the LAST phrase is specified one of the following occurs:

      1. If an exception condition is currently raised, that exception condition is set to exist in the activating runtime element. If the exception condition is an exception object, additional rules are specified in 14.6.12.1.4, Exception objects. If the exception condition is a level-3 exception for EC-USER and that exception condition is not specified in the RAISING phrase of the procedure division header of the source element in which this EXIT statement is contained, the EC-RAISING-NOT-SPECIFIED exception condition is set to exist in the activating runtime element instead of the EC-USER exception condition.

      2. If no exception condition is raised, the RAISING phrase is ignored and no exception condition is set to exist in the activating runtime element.

4) If the activating runtime element is a non-COBOL element, execution continues in the activating element in an implementor-defined fashion.

FORMAT 2

5) If the EXIT PROGRAM statement is executed in a program that is not under the control of a calling runtime element, the EXIT PROGRAM statement is treated as if it were a CONTINUE statement. No exception condition is raised even if the RAISING phrase is specified.

6) The execution of an EXIT PROGRAM statement causes the executing program to terminate, and control to return to the calling statement. If a RETURNING phrase is specified in the procedure division header of the program containing the EXIT statement, the value in the data item referenced by that RETURNING phrase becomes the result of the program activation.

   If the calling runtime element is a COBOL element, execution continues in the calling element as specified in the rules for the CALL statement. The state of the calling runtime element is not altered and is identical to that which existed at the time it executed the CALL statement except that the contents of data items and the contents of files shared between the calling runtime element and the called program may have been changed. If the program in which the EXIT PROGRAM statement is specified is an initial program, an implicit CANCEL statement referencing that program is executed upon return to the calling runtime element. That implicit

CANCEL statement will not raise any exception conditions in the calling runtime element. If the RAISING phrase is specified in the EXIT statement, the exception condition that is specified in the RAISING phrase exists after the execution of that implicit CANCEL statement.

FORMAT 3

7) The execution of an EXIT METHOD statement causes the executing method to terminate, and control to return to the invoking statement. If a RETURNING phrase is present in the containing method definition, the value in the data item referenced by the RETURNING phrase becomes the result of the method invocation.

FORMAT 4

8) The execution of an EXIT FUNCTION statement causes the executing function to terminate, and control to return to the activating statement. The value in the data item referenced by the RETURNING phrase in the containing function definition becomes the result of the function activation.

FORMAT 5

9) The execution of an EXIT PERFORM statement without the CYCLE phrase causes control to be passed to an implicit CONTINUE statement immediately following the END-PERFORM phrase that matches the most closely preceding, and as yet unterminated, inline PERFORM statement.

10) The execution of an EXIT PERFORM statement with the CYCLE phrase causes control to be passed to an implicit CONTINUE statement immediately preceding the END-PERFORM phrase that matches the most closely preceding, and as yet unterminated, inline PERFORM statement.

FORMAT 6

11) The execution of an EXIT PARAGRAPH statement causes control to be passed to an implicit CONTINUE statement immediately following the last explicit statement of the current paragraph, preceding any return mechanisms for that paragraph.

    NOTE The return mechanisms mentioned in the rules for EXIT PARAGRAPH and EXIT SECTION are those associated with language elements such as PERFORM, SORT and USE.

12) The execution of an EXIT SECTION statement causes control to be passed to an unnamed empty paragraph immediately following the last paragraph of the current section, preceding any return mechanisms for that section.

14.9.14 FREE statement

The FREE statement releases dynamic storage previously obtained with an ALLOCATE statement.

14.9.14.1 General format

FREE { data-name-1 } ...

14.9.14.2 Syntax rules

1) The data item referenced by data-name-1 shall be of category data-pointer.

14.9.14.3 General rules

1) The FREE statement is processed as follows:

    a) If the data-pointer referenced by data-name-1 identifies the start of storage that is currently allocated by an ALLOCATE statement, that storage is released and the data-pointer referenced by data-name-1 is set to NULL, the length of the released storage is the length of the storage obtained by the ALLOCATE statement, and the contents of any data items located within the released storage area become undefined;

    b) otherwise, if the data-pointer referenced by data-name-1 contains the predefined address NULL, no action is taken for that operand;

    c) otherwise, the EC-STORAGE-NOT-ALLOC exception condition is set to exist.

2) If more than one data-name-1 is specified in a FREE statement, the result of executing this FREE statement is the same as if a separate FREE statement had been written for each data-name-1 in the same order as specified in the FREE statement. If an implicit FREE statement results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, processing resumes at the next implicit FREE statement, if any.

14.9.15 GENERATE statement

The GENERATE statement performs control break processing and, unless a report-name is specified, prints one instance of the specified detail.

14.9.15.1 General format

GENERATE { data-name-1 | report-name-1 }

14.9.15.2 Syntax rules

1) Data-name-1 shall name a detail report group. It may be qualified by a report-name.

2) Report-name-1 may be used only if the referenced report description entry contains a CONTROL clause.

3) If data-name-1 is defined in a containing program, the report description entry in which data-name-1 is specified and the file description entry associated with that report description entry shall contain a GLOBAL clause.

4) If report-name-1 is defined in a containing program, the file description entry associated with report-name-1 shall contain a GLOBAL clause.

14.9.15.3 General rules

1) Execution of a GENERATE data-name statement causes one instance of the specified detail to be printed, following any necessary control break and page fit processing, as detailed below.

2) Execution of a GENERATE report-name-1 statement results in the same processing as for a GENERATE data-name statement for the same report, except that no detail is printed. This process is called summary reporting. If all the GENERATE statements executed for a report between the execution of the INITIATE and TERMINATE statements are of this form, the report that is produced is called a summary report.

3) If a CONTROL clause is specified in the report description entry, each execution of the GENERATE statement causes the specified control data items to be saved and subsequently compared so as to sense for control breaks. This processing is described by the CONTROL clause.

4) Execution of the chronologically first GENERATE statement following an INITIATE causes the following actions to take place in order:

   a) The report heading is printed if defined. If the report heading appears on a page by itself, an advance is made to the next physical page, and PAGE-COUNTER is either incremented by 1 or, if the report heading's NEXT GROUP clause has the WITH RESET phrase, set to 1. (See 13.18.56, TYPE clause; and 13.18.36, NEXT GROUP clause.)

   b) A page heading is printed if defined. (See 13.18.56, TYPE clause.)

   c) If a CONTROL clause is defined for the report, each control heading is printed, wherever defined, in order from major to minor. (See 13.18.16, CONTROL clause; and 13.18.56, TYPE clause.)

   d) The specified detail is printed, unless summary reporting is specified.

5) Execution of any chronologically second and subsequent GENERATE statements following an INITIATE causes the following actions to take place in order:

a) If a CONTROL clause is defined for the report, and a control break has been detected, each control footing and control heading is printed, if defined, up to the level of the control break. (See 13.18.16, CONTROL clause; and 13.18.56, TYPE clause.)

b) The specified detail is printed, unless summary reporting is specified.

6) If the associated report is divided into pages, the chronologically first body group since the INITIATE is preceded by a page heading, if defined, and the printing of any subsequent body groups is preceded by a page fit test that, if unsuccessful, results in a page advance, consisting of the following actions in this order:

a) If a page footing is defined it is printed.

b) An advance is made to the next physical page.

c) If the associated report description entry contains a CODE clause with an identifier operand, the identifier is evaluated.

d) If the page advance was preceded by the printing of a group whose description has a NEXT GROUP clause with the NEXT PAGE and WITH RESET phrases, PAGE-COUNTER is set to 1; otherwise PAGE-COUNTER is incremented by 1.

e) LINE-COUNTER is set to zero.

f) If a page heading is defined it is printed.

7) The report associated with data-name-1 or report-name-1 shall be in the active state. If it is not, the EC-REPORT-INACTIVE exception condition is set to exist.

8) If a nonfatal exception condition is raised during the execution of a GENERATE statement, execution resumes at the next report item, line, or report group, whichever follows in logical order.

14.9.16 GO TO statement

The GO TO statement causes control to be transferred from one part of the procedure division to another.

14.9.16.1 General format

Format 1 (unconditional):

GO TO procedure-name-1

Format 2 (depending):

GO TO { procedure-name-1 } ... DEPENDING ON identifier-1

14.9.16.2 Syntax rules

1) Identifier-1 shall reference a numeric elementary data item that is an integer.

2) If a GO TO statement represented by format 1 appears in a consecutive sequence of imperative statements within a sentence, it shall appear as the last statement in that sequence.

14.9.16.3 General rules

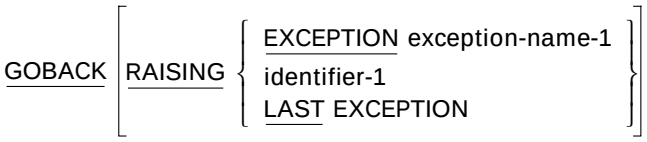
1) When a GO TO statement represented by format 1 is executed, control is transferred to procedure-name-1.

2) When a GO TO statement represented by format 2 is executed, control is transferred to procedure-name-1, etc., depending on the value of identifier-1 being 1, 2, ... , n. If the content of identifier-1 is not numeric, the EC-DATA-INCOMPATIBLE exception condition is set to exist. If checking for EC-DATA-INCOMPATIBLE is enabled, processing occurs as specified for that exception condition. If checking for EC-DATA-INCOMPATIBLE is not enabled or the value of identifier-1 is anything other than the positive or unsigned integers 1, 2, ... , n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

14.9.17 GOBACK statement

The GOBACK statement marks the logical end of a function, a method, or a program.

14.9.17.1 General format



14.9.17.2 Syntax rules

1) The GOBACK statement shall not be specified in a declarative procedure for which the GLOBAL phrase is specified in the associated USE statement.

2) Exception-name-1 shall be a level-3 exception-name as specified in 14.6.12.1, Exception conditions.

   If exception-name-1 is a level-3 exception-name for EC-USER, exception-name-1 shall be specified in the RAISING phrase of the procedure division header of the source element in which this EXIT statement is contained.

3) Identifier-1 is a sending operand.

4) Identifier-1 shall be an object reference, subject to the following constraints:

   a) If the data description entry of identifier-1 specifies a class-name, the class identified by that class-name or one of the superclasses of that class is specified in the RAISING phrase of the procedure division header of the source element containing this EXIT statement and the presence or absence of the FACTORY phrase is the same in the data description entry of identifier-1 as in the RAISING phrase of the procedure division header of the containing source element.

   b) If the data description entry of identifier-1 specifies an interface-name, the interface referenced by that interface-name shall conform to an interface specified in the RAISING phrase of the procedure division header of the source element containing this EXIT statement and the presence or absence of the FACTORY phrase is the same in the data description entry of identifier-1 as in the RAISING phrase of the procedure division header of the containing source element.

   c) If the data description entry of identifier-1 specifies an ACTIVE-CLASS phrase, the class of the object containing the EXIT statement or one of the superclasses of that class is specified in the RAISING phrase of the procedure division header of the source element containing this EXIT statement.

   d) Identifier-1 shall not be a universal object reference.

5) The LAST phrase may be specified only in a declarative procedure.

14.9.17.3 General rules

1) If a GOBACK statement is executed in a program that is under the control of a calling runtime element, the program operates as if executing an EXIT PROGRAM statement with the RAISING phrase, if any, that is specified in the GOBACK statement.

2) If a GOBACK statement is executed in a program that is not under the control of a calling runtime element, the program operates as if executing a STOP statement without any optional phrases. A RAISING phrase, if specified, is ignored.

3) If a GOBACK statement is executed in a function, the function operates as if executing an EXIT FUNCTION statement with the RAISING phrase, if any, that is specified in the GOBACK statement.

4) If a GOBACK statement is executed in a method, the method operates as if executing an EXIT METHOD statement with the RAISING phrase, if any, that is specified in the GOBACK statement.

5) If a GOBACK statement is executed within the range of a declarative procedure whose USE statement contains the GLOBAL phrase and that USE statement is specified in the same program as the GOBACK statement, the EC-FLOW-GLOBAL-GOBACK exception condition is set to exist.

14.9.18 IF statement

The IF statement causes a condition to be evaluated. The subsequent action of the runtime element depends on whether the value of the condition is true or false.

14.9.18.1 General format

Format 1 (delimited):

IF condition-1 THEN statement-1 [ELSE statement-2] END-IF

Format 2 (historic):

IF condition-1 THEN { statement-1 | NEXT SENTENCE } [ELSE { statement-2 | NEXT SENTENCE }]

NOTE NEXT SENTENCE is an archaic feature and its use should be avoided.

14.9.18.2 Syntax rules

ALL FORMATS

1) Statement-1 and statement-2 represent either one or more imperative statements or a conditional statement optionally preceded by one or more imperative statements. A further description of the rules governing statement-1 and statement-2 is given in 14.5.2, Scope of statements.

2) Statement-1 and statement-2 may each contain an IF statement. In this case, the IF statement is said to be nested. More detailed rules on nesting are given in 14.5.2, Scope of statements.

   IF statements within IF statements are considered matched IF, ELSE, and END-IF ordered combinations, proceeding from left to right. Any ELSE encountered is matched with the nearest preceding IF that either has not been already matched with an ELSE or has not been implicitly or explicitly terminated. Any END-IF encountered is matched with the nearest preceding IF that has not been implicitly or explicitly terminated.

FORMAT 2

3) The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes the terminal separator period of the sentence.

14.9.18.3 General rules

FORMAT 1

1) If condition-1 is true, control is transferred to the first statement of statement-1 and execution continues according to the rules for each statement specified in statement-1. The ELSE phrase, if specified, is ignored.

2) If condition-1 is false, the THEN phrase is ignored. If the ELSE phrase is specified, control is transferred to the first statement of statement-2 and execution continues according to the rules for each statement specified in statement-2.

FORMAT 2

3) If condition-1 is true and statement-1 is specified, control is transferred to the first statement of statement-1 and execution continues according to the rules for each statement specified in statement-1. The ELSE phrase, if specified, is ignored.

4) If condition-1 is true and NEXT SENTENCE is specified in the THEN phrase, the ELSE phrase, if specified, is ignored and control is transferred to an implicit CONTINUE statement immediately preceding the next separator period.

5) If condition-1 is false and statement-2 is specified, the THEN phrase is ignored, control is transferred to the first statement of statement-2, and execution continues according to the rules for each statement specified in statement-2.

6) If condition-1 is false and NEXT SENTENCE is specified in the ELSE phrase, the THEN phrase is ignored and control is transferred to an implicit CONTINUE statement immediately preceding the next separator period.

7) If condition-1 is false and the ELSE phrase is not specified, the THEN phrase is ignored.

14.9.19 INITIALIZE statement

The INITIALIZE statement provides the ability to set selected data items to specified values.

14.9.19.1 General format

```
INITIALIZE { identifier-1 } ... [ WITH FILLER ]
    [ { ALL           } TO VALUE ]
    [ { category-name }          ]
    [ THEN REPLACING { category-name DATA BY { identifier-2 } } ... ]
                                             { literal-1    }
    [ THEN TO DEFAULT ]
```

where category-name is:

```
{ ALPHABETIC           }
{ ALPHANUMERIC         }
{ ALPHANUMERIC-EDITED  }
{ BOOLEAN              }
{ DATA-POINTER         }
{ FUNCTION-POINTER     }
{ NATIONAL             }
{ NATIONAL-EDITED      }
{ NUMERIC              }
{ NUMERIC-EDITED       }
{ OBJECT-REFERENCE     }
{ PROGRAM-POINTER      }
```

14.9.19.2 Syntax rules

1) Identifier-1 shall be strongly typed or of class alphabetic, alphanumeric, boolean, national, numeric, object, or pointer.

2) For each DATA-POINTER, FUNCTION-POINTER, OBJECT-REFERENCE, or PROGRAM-POINTER phrase specified as the category-name in the REPLACING phrase, identifier-2 shall be specified.

3) For each of the categories data-pointer, function-pointer, object-reference, and program-pointer specified in the REPLACING phrase, a SET statement with identifier-2 as the sending operand and an item of the specified category as the receiving operand shall be valid.

   For each of the other categories specified in the REPLACING phrase, a MOVE statement with identifier-2 or literal-1 as the sending item and an item of the specified category as the receiving operand shall be valid.

4) The data description entry for the data item referenced by identifier-1 shall not contain a RENAMES clause.

5) The same category shall not be repeated in a REPLACING phrase.

6) The data item referenced by identifier-1 is the receiving operand.

7) If the REPLACING phrase is specified, literal-1 or the data item referenced by identifier-2 is the sending operand. If the REPLACING phrase is not specified, the sending operand is determined according to the general rules of the INITIALIZE statement.

14.9.19.3 General rules

1) When identifier-1 references a bit group item or a national group item, identifier-1 is processed as a group item. When identifier-2 references a bit group item or a national group item, identifier-2 is processed as an elementary data item.

2) The keywords in category-name correspond to a category of data as specified in 8.5.2, Class and category of data items and literals. If ALL is specified in the VALUE phrase it is as if all of the categories listed in category-name were specified.

3) If more than one identifier-1 is specified in an INITIALIZE statement, the result of executing this INITIALIZE statement is the same as if a separate INITIALIZE statement had been written for each identifier-1 in the same order as specified in the INITIALIZE statement. If an implicit INITIALIZE statement results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, processing resumes at the next implicit INITIALIZE statement, if any.

4) Whether identifier-1 references an elementary item or a group item, the effect of the execution of the INITIALIZE statement is as though a series of implicit MOVE or SET statements, each of which has an elementary data item as its receiving operand, were executed. The sending-operands of these implicit statements are defined in general rule 6 and the receiving-operands are defined in general rule 5.

   If the category of a receiving-operand is data-pointer, function-pointer, object-reference, or program-pointer, the implicit statement is:

   SET receiving-operand TO sending-operand

   Otherwise, the implicit statement is:

   MOVE sending-operand TO receiving-operand.

5) The receiving-operand in each implicit MOVE or SET statement is determined by applying the following steps in order:

   a) First, the following data items are excluded as receiving-operands:

      1. Any identifiers that are not valid receiving operands of a MOVE statement, except data items of category data-pointer, object-reference, or program-pointer.

      2. If the FILLER phrase is not specified, elementary data items with an explicit or implicit FILLER clause.

      3. Any elementary data item subordinate to identifier-1 whose data description entry contains a REDEFINES or RENAMES clause or is subordinate to a data item whose data description entry contains a REDEFINES clause. However, identifier-1 may itself have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause.

   b) Second, an elementary data item is a possible receiving item if:

      1. It is explicitly referenced by identifier-1; or

      2. It is contained within the group data item referenced by identifier-1. If the elementary data item is a table element, each occurrence of the elementary data item is a possible receiving-operand.

c) Finally, each possible receiving-operand is a receiving-operand if at least one of the following is true:

1. The VALUE phrase is specified, the category of the elementary data item is one of the categories specified or implied in the VALUE phrase, and one of the following is true:

   a. Either the category of the elementary data item is data-pointer, object-reference, or program-pointer, or

   b. A data-item format VALUE clause is specified in the data description entry of the elementary data item.

   c. A table format VALUE clause is specified in the data description entry of the elementary item and that VALUE clause specifies a value for the particular occurrence of the elementary data item.

2. The REPLACING phrase is specified and the category of the elementary data item is one of the categories specified in the REPLACING phrase; or

3. The DEFAULT phrase is specified; or

4. Neither the REPLACING phrase nor the VALUE phrase is specified.

6) The sending-operand in each implicit MOVE and SET statement is determined as follows:

a) If the data item qualifies as a receiving-operand because of the VALUE phrase:

1. If the category of the receiving-operand is data-pointer, function-pointer, or program-pointer, the sending-operand is the predefined address NULL;

2. If the category of the receiving-operand is object-reference, the sending-operand is the predefined object reference NULL;

3. Otherwise, the sending-operand is determined by the literal in the VALUE clause specified in the data description entry of the data item. If the data item is a table element, the literal in the VALUE clause that corresponds to the occurrence being initialized determines the sending-operand. The actual sending-operand is a literal that, when moved to the receiving-operand with a MOVE statement, produces the same result as the initial value of the data item as produced by the application of the VALUE clause.

b) If the data item does not qualify as a receiving-operand because of the VALUE phrase, but does qualify because of the REPLACING phrase, the sending-operand is the literal-1 or identifier-2 associated with the category specified in the REPLACING phrase.

c) If the data item does not qualify in accordance with general rules 6a and 6b, the sending-operand used depends on the category of the receiving-operand as follows:

Receiving operand	**Figurative constant**
Alphabetic	Figurative constant alphanumeric SPACES
Alphanumeric	Figurative constant alphanumeric SPACES
Alphanumeric-edited	Figurative constant alphanumeric SPACES
Boolean	Figurative constant ZEROES
Data-pointer	Predefined address NULL
Function-pointer	Predefined address NULL
National	Figurative constant national SPACES
National-edited	Figurative constant national SPACES
Numeric	Figurative constant ZEROES
Numeric-edited	Figurative constant ZEROES
Object-reference	Predefined object reference NULL

Program-pointer	Predefined address NULL

NOTE When an any-length elementary item is initialized, its length is set to zero.

7) If identifier-1 references a group data item, affected elementary data items are initialized in the sequence of their definition within the group data item. For a variable-occurrence data item, the number of occurrences initialized is determined by the rules of the OCCURS clause for a receiving data item.

8) If identifier-1 occupies the same storage area as identifier-2, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See 14.6.9, Overlapping operands.)

9) When a group containing a dynamic-capacity table is initialized, all the entries of the table up to current capacity, if any, are initialized, whether or not the INITIALIZED phrase is present in the OCCURS clause, and the current capacity of the table is left unchanged.

14.9.20 INITIATE statement

The INITIATE statement initializes any internal storage locations used by the specified reports before any printing begins for these reports.

14.9.20.1 General format

INITIATE { report-name-1 } ...

14.9.20.2 Syntax rules

1) Report-name-1 shall be defined by a report description entry in the report section.

2) If report-name-1 is defined in a containing program, the file description entry associated with report-name-1 shall contain a GLOBAL clause.

14.9.20.3 General rules

1) The INITIATE statement performs the following initialization functions for each specified report:

   a) All sum counters and all size error indicators are set to zero.

   b) LINE-COUNTER is set to zero.

   c) PAGE-COUNTER is set to 1.

2) An INITIATE statement shall not be executed if report-name-1 is in the active state. If it is in the active state, the EC-REPORT-ACTIVE exception condition is set to exist and the execution of the INITIATE statement has no other effect.

3) The INITIATE statement does not open any file connector with which report-name-1 is associated. Therefore, the INITIATE statement may be executed only if the corresponding file connector is open in the extend mode or the output mode. If the file connector is not open in the output or extend mode, the EC-REPORT-FILE-MODE exception condition is set to exist and no action is taken on the report.

4) A successful INITIATE statement places the report in the active state.

5) The results of executing an INITIATE statement in which more than one report-name-1 is specified is the same as if a separate INITIATE statement had been executed for each report-name-1 in the same order as specified in the statement. If an implicit INITIATE statement results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, processing resumes at the next implicit INITIATE statement, if any.

14.9.21 INSPECT statement

The INSPECT statement provides the ability to tally or replace occurrences of single characters or sequences of characters in a data item.

14.9.21.1 General format

Format 1 (tallying):

INSPECT identifier-1 TALLYING tallying-phrase

Format 2 (replacing):

INSPECT identifier-1 REPLACING replacing-phrase

Format 3 (tallying-and-replacing):

INSPECT identifier-1 TALLYING tallying-phrase REPLACING replacing-phrase

Format 4 (converting):

INSPECT identifier-1 CONVERTING { identifier-6 / literal-4 } TO { identifier-7 / literal-5 } [after-before-phrase]

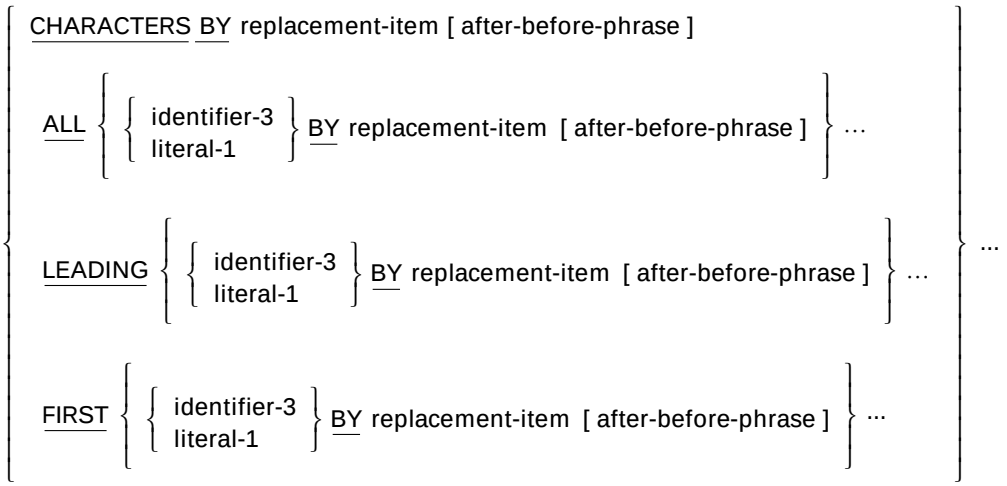
where tallying-phrase is:

{ identifier-2 FOR { CHARACTERS [after-before-phrase] / ALL { { identifier-3 / literal-1 } [after-before-phrase] } ... / LEADING { { identifier-3 / literal-1 } [after-before-phrase] } ... } ... } ...

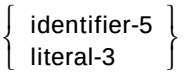
where after-before-phrase is:

{ AFTER INITIAL { identifier-4 / literal-2 } / BEFORE INITIAL { identifier-4 / literal-2 } }

where replacing-phrase is:



where replacement-item is:



14.9.21.2 Syntax rules

ALL FORMATS

1) Identifier-1 shall reference either an alphanumeric or national group item or an elementary item described implicitly or explicitly as usage display or national.

2) Identifier-3, ... , identifier-n shall reference an elementary item described implicitly or explicitly as usage display or national.

3) Each literal shall be an alphanumeric, boolean, or national literal. Literal-1, literal-2, literal-3, literal-4 shall not be a figurative constant that begins with the word ALL. If literal-1, literal-2, or literal-4 is a figurative constant, it refers to an implicit one character data item. When identifier-1 is of class national, the class of the figurative constant is national; when identifier-1 is of class boolean, the figurative constant is of class boolean and only the figurative constant ZERO may be specified; otherwise, the class of the figurative constant is alphanumeric. Literal-1, literal-2, literal-3, literal-4 or literal-5 shall not be a zero-length literal.

4) If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, identifier-7, literal-1, literal-2, literal-3, literal-4, or literal-5 references an elementary data item or literal of class boolean or national, then all shall reference a data item or literal of class boolean or national, respectively.

FORMATS 1 AND 3

5) Identifier-2 shall reference an elementary numeric data item.

FORMATS 2 AND 3

6) When both literal-1 and literal-3 are specified, they shall be the same size except when literal-3 is a figurative constant, in which case it is expanded or contracted to be the size of literal-1.

7) When the CHARACTERS phrase is specified, literal-3 shall be one character in length.

FORMAT 1

8) Identifier-1 is a sending operand.

FORMAT 4

9) When both literal-4 and literal-5 are specified they shall be the same size except when literal-5 is a figurative constant.

14.9.21.3 General rules

ALL FORMATS

1) For purposes of determining its length, identifier-1 is treated as a sending data item.

2) If the data item referenced by identifier-1 is a zero-length item:

   a) the contents of the data items referenced by identifier-1 and identifier-2 are unchanged

   b) the execution of the INSPECT statement is successful

   c) control is immediately transferred to the end of the INSPECT statement.

3) Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position as described in general rules 6 through 8.

4) For use in the INSPECT statement, the content of the data item referenced by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 shall be treated as follows:

   a) If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 references an alphabetic, alphanumeric, boolean, or national data item, the INSPECT statement shall treat the content of each such identifier as a character-string of the category associated with that identifier.

   b) If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 references an alphanumeric-edited data item, or a numeric-edited or unsigned numeric data item described explicitly or implicitly with usage display, the data item is inspected as though it had been redefined as alphanumeric (see general rule 4a) and the INSPECT statement had been written to reference the redefined data item.

   c) If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 references a national-edited data item, or a numeric-edited or unsigned numeric data item described explicitly or implicitly with usage national, the data item is inspected as though it had been redefined as category national and the INSPECT statement has been written to reference the redefined data item.

   d) If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 references a signed numeric data item, the data item is inspected as though it had been moved to an unsigned numeric data item with length equal to the length of the signed item excluding any separate sign position, and then the rules in general rule 4b or 4c had been applied. If identifier-1 is a signed numeric item, the original value of the sign is retained upon completion of the INSPECT statement.

5) In general rules 6 through 21, all references to literal-1, literal-2, literal-3, literal-4, or literal-5 apply equally to the content of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 respectively.

6) Item identification for any identifier is done only once as the first operation in the execution of the INSPECT statement.

FORMATS 1 AND 2

7) During inspection of the content of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (format 1) or replaced by literal-3 (format 2).

8) The comparison operation to determine the occurrence of literal-1 to be tallied or to be replaced, occurs as follows:

   a) The operands of the TALLYING or REPLACING phrase are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1 is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by identifier-1. Literal-1 matches that portion of the content of the data item referenced by identifier-1 if they are equal, character for character and:

      1. If neither LEADING nor FIRST is specified; or

      2. If the LEADING adjective applies to literal-1 and literal-1 is a leading occurrence as defined in general rules 12 and 17; or

      3. If the FIRST adjective applies to literal-1 and literal-1 is the first occurrence as defined in general rule 17.

   b) If no match occurs in the comparison of the first literal-1, the comparison is repeated with each successive literal-1, if any, until either a match is found or there is no next successive literal-1. When there is no next successive literal-1, the character position in the data item referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1.

   c) Whenever a match occurs, tallying or replacing takes place as described in general rules 12 and 15. The character position in the data item referenced by identifier-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with the first literal-1.

   d) The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

   e) If the CHARACTERS phrase is specified, an implied one character operand participates in the cycle described in general rules 8a through 8d above as if it had been specified by literal-1, except that no comparison to the content of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the content of the data item referenced by identifier-1 participating in the current comparison cycle.

9) The comparison operation defined in general rule 8 is restricted by the BEFORE and AFTER phrase as follows:

   a) If neither the BEFORE nor AFTER phrase is specified or identifier-4 references a zero-length item, literal-1 or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in general rule 8. Literal-1 or the implied operand of the CHARACTERS phrase is first eligible to participate in matching at the leftmost character position of identifier-1.

   b) If the BEFORE phrase is specified, the associated literal-1 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles that involve that portion of the content of the data item referenced by identifier-1 from its leftmost character position up to, but not including, the first occurrence of literal-2 within the content of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 8 is begun. If, on any comparison cycle, literal-1 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the content of the data item referenced by identifier-1. If there is no occurrence of literal-2 within the content of the data item referenced by identifier-1, its

associated literal-1 or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

c) If the AFTER phrase is specified, the associated literal-1 or the implied operand of the CHARACTERS phrase participate only in those comparison cycles that involve that portion of the content of the data item referenced by identifier-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2 within the content of the data item referenced by identifier-1 to the rightmost character position of the data item referenced by identifier-1. This is the character position at which literal-1 or the implied operand of the CHARACTERS phrase is first eligible to participate in matching. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 8 is begun. If, on any comparison cycle, literal-1 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the content of the data item referenced by identifier-1. If there is no occurrence of literal-2 within the content of the data item referenced by identifier-1, its associated literal-1 or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

FORMAT 1

10) Both the ALL and LEADING phrases are transitive across the operands that follow them until another ALL or LEADING phrase is encountered.

11) The content of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.

12) The rules for tallying are as follows:

a) If the ALL phrase is specified, the content of the data item referenced by identifier-2 is incremented by one for each occurrence of literal-1 matched within the content of the data item referenced by identifier-1.

b) If the LEADING phrase is specified, the content of the data item referenced by identifier-2 is incremented by one for the first and each subsequent contiguous occurrence of literal-1 matched within the content of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

c) If the CHARACTERS phrase is specified, the content of the data item referenced by identifier-2 is incremented by one for each character matched, in the sense of general rule 8e, within the content of the data item referenced by identifier-1.

13) If identifier-1, identifier-3, or identifier-4 occupies the same storage area as identifier-2, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See 14.6.9, Overlapping operands.)

FORMATS 2 AND 3

14) The size of literal-3 or the data item referenced by identifier-5 shall be equal to the size of literal-1 or the data item referenced by identifier-3. If these sizes are not equal, the EC-RANGE-INSPECT-SIZE exception condition is set to exist and the results of the execution of the INSPECT statement are undefined. When a figurative constant is used as literal-3, the size of the figurative constant is equal to the size of literal-1 or the size of the data item referenced by identifier-3.

15) When the CHARACTERS phrase is used, the data item referenced by identifier-5 shall be one character in length. If it is not, the EC-RANGE-INSPECT-SIZE exception condition is set to exist and the results of the execution of the INSPECT statement are undefined.

FORMAT 2

16) The ALL, FIRST, and LEADING phrases are transitive across the operands that follow them until another ALL, FIRST, or LEADING phrase is encountered.

17) The rules for replacement are as follows:

    a) When the CHARACTERS phrase is specified, each character matched, in the sense of general rule 8e, in the content of the data item referenced by identifier-1 is replaced by literal-3.

    b) When the adjective ALL is specified, each occurrence of literal-1 matched in the content of the data item referenced by identifier-1 is replaced by literal-3.

    c) When the adjective LEADING is specified, the first and each successive contiguous occurrence of literal-1 matched in the content of the data item referenced by identifier-1 is replaced by literal-3, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

    d) When the adjective FIRST is specified, the leftmost occurrence of literal-1 matched within the content of the data item referenced by identifier-1 is replaced by literal-3. This rule applies to each successive specification of the FIRST phrase regardless of the value of literal-1.

18) If identifier-3, identifier-4, or identifier-5 occupies the same storage area as identifier-1, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See 14.6.9, Overlapping operands.)

FORMAT 3

19) A format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being a format 1 statement with TALLYING phrases identical to those specified in the format 3 statement, and the other statement being a format 2 statement with REPLACING phrases identical to those specified in the format 3 statement. The general rules given for matching and counting apply to the format 1 statement and the general rules given for matching and replacing apply to the format 2 statement. Item identification of any identifier in the format 2 statement is done only once before executing the format 1 statement.

FORMAT 4

20) A format 4 INSPECT statement is interpreted and executed as though a format 2 INSPECT statement specifying the same identifier-1 had been written with a series of ALL phrases, one for each character of literal-4. The effect is as if each of these ALL phrases referenced, as literal-1, a single character of literal-4 and referenced, as literal-3, the corresponding single character of literal-5. Correspondence between the characters of literal-4 and the characters of literal-5 is by ordinal position within the data item.

21) If identifier-4, identifier-6, or identifier-7 occupies the same storage area as identifier-1, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See 14.6.9, Overlapping operands.)

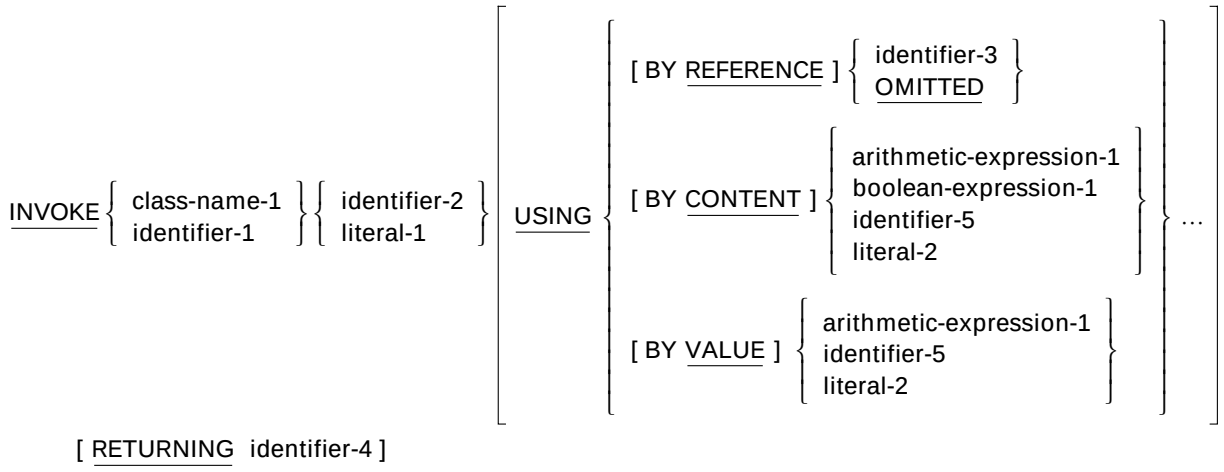
22) The size of literal-5 or the data item referenced by identifier-7 shall be equal to the size of literal-4 or the data item referenced by identifier-6. If these sizes are not equal, the EC-RANGE-INSPECT-SIZE exception condition is set to exist and the results of the execution of the INSPECT statement are undefined. When a figurative constant is used as literal-5, the size of the figurative constant is equal to the size of literal-4 or the size of the data item referenced by identifier-6.

23) If the same character appears more than once in the data item referenced by identifier-6 or in literal-4, the first occurrence of the character is used for replacement.

14.9.22 INVOKE statement

The INVOKE statement causes a method to be invoked.

14.9.22.1 General format



14.9.22.2 Syntax Rules

1) Identifier-1 shall be an object reference.

2) Literal-1 shall be of class alphanumeric or national and shall not be a zero-length literal.

3) If class-name-1 is specified, literal-1 shall be specified. The value of literal-1 shall be the name of a method defined in the factory interface of class-name-1.

4) If identifier-1 is specified and it does not reference a universal object reference, literal-1 shall be specified. The value of literal-1 shall be the name of a method, subject to the following conditions:

   a) If identifier-1 references an object reference described with a class-name and the FACTORY phrase, literal-1 shall be the name of a method contained in the factory interface of that class-name.

   b) If identifier-1 references an object reference described with a class-name without the FACTORY phrase, literal-1 shall be the name of a method contained in the instance interface of that class-name.

   c) If identifier-1 references an object reference described with the ACTIVE-CLASS and the FACTORY phrases, literal-1 shall be the name of a method contained in the factory interface of the class containing the INVOKE statement.

   d) If identifier-1 references an object reference described with the ACTIVE-CLASS phrase and without the FACTORY phrase, literal-1 shall be the name of a method contained in the instance interface of the class containing the INVOKE statement.

   e) If identifier-1 references an object reference described with an interface-name, literal-1 shall be the name of a method contained in the interface referenced by that interface-name.

   f) If identifier-1 references the predefined object reference SELF and the method containing the INVOKE statement is a factory method, literal-1 shall be the name of a method contained in the factory interface of the class containing the INVOKE statement.

g) If identifier-1 references the predefined object reference SELF and the method containing the INVOKE statement is an instance method, literal-1 shall be the name of a method contained in the instance interface of the class containing the INVOKE statement.

h) If identifier-1 references the predefined object reference SUPER and the method containing the INVOKE statement is a factory method, literal-1 shall be the name of a method contained in the factory interface of a class inherited by the class containing the INVOKE statement.

i) If identifier-1 references the predefined object reference SUPER and the method containing the INVOKE statement is an instance method, literal-1 shall be the name of a method contained in the instance interface of a class inherited by the class containing the INVOKE statement.

5) If class-name-1 is specified or the data item referenced by identifier-1 is not a universal object reference, the following applies:

a) If a BY CONTENT or BY REFERENCE phrase is specified for an argument, a BY REFERENCE phrase shall be specified or implied for the corresponding formal parameter in the procedure division header.

b) If a BY VALUE phrase is specified for an argument, a BY VALUE phrase shall be specified for the corresponding formal parameter in the procedure division header.

c) The rules for conformance as specified in 14.8, Conformance for parameters and returning items, apply.

6) If identifier-1 references a universal object reference, neither the BY CONTENT nor the BY VALUE phrase shall be specified and the BY REFERENCE phrase, if not specified explicitly, is assumed implicitly.

7) Identifier-2 may be specified only when identifier-1 is a universal object reference.

8) Identifier-2 shall reference an alphanumeric or national data item.

9) Identifier-3 shall be an address-identifier or shall reference a data item defined in the file, working-storage, local-storage, or linkage section.

10) Identifier-3 shall not reference a data item defined in the file or working-storage section of a factory or instance object.

11) Identifier-4 shall reference a data item defined in the file, working-storage, local-storage, or linkage section.

12) If identifier-3 or identifier-4 references a bit data item, it shall be described such that:

a) subscripting and reference modification in that identifier consist of only fixed-point numeric literals or arithmetic expressions in which all operands are fixed-point numeric literals and the exponentiation operator is not specified; and

b) it is aligned on a byte boundary.

13) If Identifier-3, identifier-4, or identifier-5 references a group item, there shall not be an item subordinate to that group item that is an object reference described with the ACTIVE-CLASS phrase.

14) If an argument is specified without any BY phrase, BY REFERENCE is implied for that argument when:

a) that argument is valid for being passed by reference, and

b) the corresponding formal parameter is not specified with the BY VALUE phrase.

15) If identifier-5 or its corresponding formal parameter is specified with the BY VALUE phrase, identifier-5 shall be of class numeric, object or pointer.

16) If literal-2 or its corresponding formal parameter is specified with the BY VALUE phrase, literal-2 shall be a numeric literal.

17) Literal-2 shall not be a zero-length literal.

18) If an OMITTED phrase is specified, an OPTIONAL phrase shall be specified for the corresponding formal parameter in the procedure division header.

19) If identifier-3 references an address-identifier, identifier-3 is a sending operand.

20) If identifier-3 does not reference an address-identifier, identifier-3 is a receiving operand.

21) Identifier-5 and any identifier specified in arithmetic-expression-1 or boolean-expression-1 is a sending operand.

22) Identifier-4 is a receiving operand.

14.9.22.3 General Rules

1) The instance of the program, function, or method that executes the INVOKE statement is the activating runtime element.

2) Identifier-1 identifies an instance object. If class-name-1 is specified, it identifies the factory object of the class referenced by that class-name. Literal-1 or the content of the data item referenced by identifier-2 identifies a method of that object that will act upon that instance object:

   a) If the method to be invoked is a COBOL method, literal-1 or the content of the data item referenced by identifier-2 is the name of the method to be invoked as described in 8.3.1.1.1, User-defined words.

   b) If the method to be invoked is a non-COBOL method, the behavior of the INVOKE statement is implementor-defined.

3) The sequence of arguments in the USING phrase of the INVOKE statement and the corresponding formal parameters in the USING phrase of the invoked method's procedure division header determines the correspondence between arguments and formal parameters. This correspondence is positional and not by name equivalence.

   NOTE The first argument corresponds to the first formal parameter, the second to the second, and the nth to the nth.

   The effect of the USING phrase on the activated runtime element is described in 14.3, Procedure division, general rules.

4) An argument that consists merely of a single identifier or literal is regarded as an identifier or literal rather than an arithmetic or boolean expression.

5) If identifier-1 is null, the EC-OO-NULL exception condition is set to exist and execution of the INVOKE statement is terminated.

6) If class-name-1 is specified or the data item referenced by identifier-1 is not a universal object reference, the following applies: If an argument is specified without any of the keywords BY REFERENCE, BY CONTENT, or BY VALUE, the manner used for passing that argument is determined as follows:

   a) When the BY REFERENCE phrase is specified or implied for the corresponding formal parameter:

      1. if the argument meets the requirements of syntax rules 9 and 10, BY REFERENCE is assumed;

      2. if the argument does not meet the requirements of syntax rules 9 and 10, BY CONTENT is assumed.

b) When the BY VALUE phrase is specified or implied for the corresponding formal parameter, BY VALUE is assumed.

7) Execution of the INVOKE statement proceeds as follows:

a) Arithmetic-expression-1, boolean-expression-1, identifier-1, identifier-2, identifier-3, and identifier-5 are evaluated and item identification is done for identifier-4 at the beginning of the execution of the INVOKE statement. If an exception condition exists, no method is invoked and execution proceeds as specified in general rule 7f. If an exception condition does not exist, the values of identifier-3, identifier-5, arithmetic-expression-1, boolean-expression-1, or literal-2 are made available to the invoked method at the time control is transferred to that method.

b) The runtime system attempts to locate the method being invoked using the rules specified in 8.4.5, Scope of names; 8.4.5.4, Scope of method-names; 9.3.6, Method invocation; and 12.3.7, REPOSITORY paragraph. If the method is not found or the resources necessary to execute the method are not available, the EC-OO-METHOD exception condition is set to exist, the method is not activated, and execution continues as specified in general rule 7f.

c) If identifier-1 is a universal object reference and the method being invoked is a COBOL method, neither a formal parameter nor the returning item in the invoked method shall be described with the ANY LENGTH clause, and the rules for conformance specified in 14.8, Conformance for parameters and returning items, apply. If a violation of these rules is detected, the EC-OO-UNIVERSAL exception condition is set to exist if checking for it is enabled in both the invoked method and the activating runtime element, and execution continues as specified in general rule 7f.

d) The method specified by the INVOKE statement is made available for execution and control is transferred to the invoked method. Control is transferred to the invoked method in a manner consistent with the entry convention specified for the method. If the invoked method is a COBOL method, its execution is described in 14.2.3, General rules of the procedure division; otherwise, the execution is defined by the implementor.

e) After control is returned from the invoked method, if an exception condition is propagated from the invoked method, execution continues as specified in general rule 7f; otherwise, control is transferred to the end of the INVOKE statement.

f) If an exception condition has been raised, any declarative that is associated with that exception condition is executed. Execution then proceeds as defined for the exception condition and execution of the declarative.

8) If a RETURNING phrase is specified, the result of the activated method is placed into identifier-4.

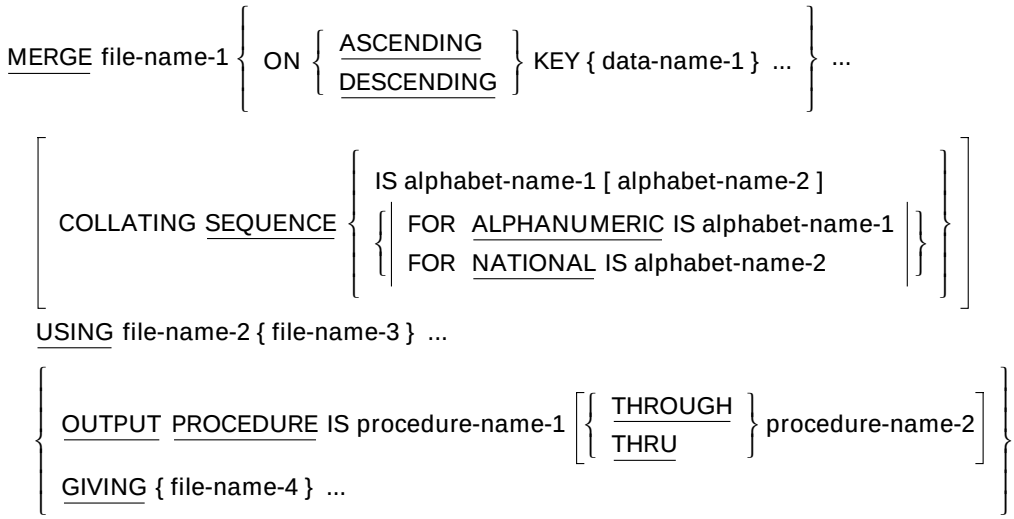
9) If an OMITTED phrase is specified or a trailing argument is omitted, the omitted-argument condition for that parameter shall be true in the invoked method.

10) If a parameter for which the omitted-argument condition is true is referenced in an invoked method, except as an argument or in the omitted-argument condition, the EC-PROGRAM-ARG-OMITTED exception condition is set to exist.

14.9.23 MERGE statement

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merged order, to an output procedure or to an output file.

14.9.23.1 General format



14.9.23.2 Syntax rules

1) A MERGE statement may appear anywhere in the procedure division except in the declaratives portion.

2) File-name-1 shall be described in a sort-merge file description entry in the data division.

3) If the file description entry for file-name-1 describes variable-length records, the file description entry for file-name-2 or file-name-3 shall describe neither records smaller than the smallest record nor larger than the largest record described for file-name-1. If the file description entry for file-name-1 describes fixed-length records, the file description entry for file-name-2 or file-name-3 shall not describe a record that is larger than the record described for file-name-1.

4) Data-name-1 is a key data-name. Key data-names are subject to the following rules:

   a) The data items identified by key data-names shall be described in records associated with file-name-1.

   b) Key data-names may be qualified.

   c) Key data items shall not be of the class boolean, object, or pointer.

   d) A key data item shall not be a variable-length group, an occurs-depending-on data item, an any-length elementary item, or an item subordinate to a dynamic-capacity table.

   e) If file-name-1 has more than one record description, the data items identified by key data-names need be described in only one of the record descriptions. The same byte positions referenced by a key data-name in one record description entry are taken as the key in all records of the file.

   f) None of the data items identified by key data-names may be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

g) If the file referenced by file-name-1 contains variable-length records, all the data items identified by key data-names shall be contained within the first x bytes of the record, where x equals the minimum record size specified for the file referenced by file-name-1.

5) Alphabet-name-1 shall reference an alphabet that defines an alphanumeric collating sequence.

6) Alphabet-name-2 shall reference an alphabet that defines a national collating sequence.

7) File-names shall not be repeated within the MERGE statement.

8) The words THROUGH and THRU are equivalent.

9) File-name-2, file-name-3, and file-name-4 shall be described in a file description entry that is not for a report file and is not a sort-merge file description entry.

10) If file-name-4 references an indexed file, the first specification of data-name-1 shall be associated with an ASCENDING phrase and the data item referenced by that data-name-1 shall occupy the same byte positions in its record as the data item associated with the prime record key for that file.

11) No pair of file-names in a MERGE statement may be specified in the same SAME AREA, SAME SORT AREA, or SAME SORT-MERGE AREA clause. The only file-names in a MERGE statement that may be specified in the same SAME RECORD AREA clause are those associated with the GIVING phrase.

12) If the GIVING phrase is specified and the file description entry for file-name-4 describes variable-length records, the file description entry for file-name-1 shall describe neither records smaller than the smallest record nor larger than the largest record described for file-name-4. If the file description entry for file-name-4 describes fixed-length records, the file description entry for file-name-1 shall not describe a record that is larger than the record described for file-name-4.

13) If file-name-2, or file-name-3 references a relative or an indexed file, its access mode shall be sequential or dynamic.

14.9.23.3 General rules

1) The MERGE statement merges all records contained on the files referenced by file-name-2 and file-name-3.

2) If the file referenced by file-name-1 contains only fixed-length records, any record in the file referenced by file-name-2 or file-name-3 containing fewer character positions than that fixed length is space filled on the right to that fixed length, beginning with the first character position after the last character in the record, when that record is released to the file referenced by file-name-1, as follows:

   a) If there is only one record description entry associated with the file referenced by file-name-2 or file-name-3 and that record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.

   b) If there are multiple record description entries associated with the file referenced by file-name-2 or file-name-3 and the descriptions include a SELECT WHEN clause, the rules of the SELECT WHEN clause are applied to the record to select its description. When the record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.

   c) Otherwise, the record is filled with alphanumeric space characters.

3) The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. The leftmost data-name is the major key, the next data-name is the next most significant key, etc.

a) When the ASCENDING phrase is specified, the merged sequence is from the lowest value of the contents of the data items identified by the key data-names to the highest value, according to the rules for comparison of operands in a relation condition.

b) When the DESCENDING phrase is specified, the merged sequence is from the highest value of the contents of the data items identified by the key data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

The words ASCENDING and DESCENDING are transitive across all occurrences of data-name-1 until another occurrence of the word ASCENDING or DESCENDING is encountered.

4) When, according to the rules for the comparison of operands in a relation condition, the contents of all the key data items of one record are equal to the contents of the corresponding key data items of one or more other records, the order of return of these records:

a) Follows the order of the associated input files as specified in the MERGE statement.

b) Is such that all records associated with one input file are returned prior to the return of records from another input file.

5) The alphanumeric collating sequence that applies to the comparison of key data items of class alphabetic and class alphanumeric, and the national collating sequence that applies to the comparison of key data items of class national, are separately determined at the beginning of the execution of the MERGE statement in the following order of precedence:

a) First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in this MERGE statement. The collating sequence associated with alphabet-name-1 applies to key data items of class alphabetic and alphanumeric; the collating sequence associated with alphabet-name-2 applies to key data items of class national.

b) Second, the collating sequences established as the program collating sequences.

6) If the records in the file referenced by file-name-2 and file-name-3 are not ordered as described in the ASCENDING or DESCENDING KEY clauses and the collating sequence associated with the MERGE statement, the EC-SORT-MERGE-SEQUENCE exception condition is set to exist, all files associated with the MERGE statement are closed, and the results of the merge operation are undefined.

7) All the records in the files referenced by file-name-2 and file-name-3 are transferred to the file referenced by file-name-1. At the start of execution of the MERGE statement, the file connectors referenced by file-name-2 and file-name-3 shall not be in the open mode, otherwise the EC-SORT-MERGE-FILE-OPEN exception condition is set to exist and the execution of the MERGE statement terminates. For each of the files referenced by file-name-2 and file-name-3 the execution of the MERGE statement causes the following actions to be taken:

a) The processing of the file is initiated. If the file-control entry for the file has a SHARING clause with the ALL phrase, the initiation is performed as if an OPEN statement with the INPUT phrase and the SHARING WITH READ ONLY phrase had been executed; otherwise, the initiation is performed as if an OPEN statement with the INPUT phrase and without a SHARING phrase is executed. If an output procedure is specified, this initiation is performed before control passes to the output procedure. If a nonfatal exception condition exists as a result of the execution of the implicit OPEN statement, the MERGE statement is terminated unless there is an applicable USE procedure that completes normally, after which the MERGE statement continues processing as if the exception condition did not exist.

b) The logical records are obtained and released to the merge operation. Each record is obtained as if a READ statement with the NEXT phrase, the IGNORING LOCK phrase, and the AT END phrase had been executed. When the at end condition exists for file-name-2 or file-name-3, the processing for that file connector is terminated. If the file referenced by file-name-1 is described with variable-length records, the size of any record written to file-name-1 is the size of that record when it was read from file-name-2 or file-name-3, regardless of the content of the data item referenced by the DEPENDING ON phrase of either a RECORD

IS VARYING clause or an OCCURS clause specified in the sort-merge file description entry for file-name-1. If the size of the record read from the file referenced by file-name-2 or file-name-3 is larger than the largest record allowed in the file description entry for file-name-1, the EC-SORT-MERGE-RELEASE exception condition is set to exist and the execution of the MERGE statement is terminated. If file-name-1 is specified with variable-length records and the size of the record read from the file referenced by file-name-2 or file-name-3 is smaller than the smallest record allowed in the file description entry for file-name-1, the EC-SORT-MERGE-RELEASE exception condition is set to exist and the execution of the MERGE statement is terminated.

c) The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. If an output procedure is specified, this termination is not performed until after control passes the last statement in the output procedure. For a relative file, the content of the relative key data item is undefined after the execution of the MERGE statement.

These implicit functions are performed such that any applicable USE procedures are executed. If a nonfatal exception condition exists from an attempt to CLOSE file-name-2 or file-name-3:

1. If there is an applicable USE procedure that completes normally, the MERGE statement continues.

2. If there is no applicable USE procedure, the exception condition is ignored and the MERGE statement continues execution.

The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the file description entry for file-name-2 or file-name-3 is undefined upon completion of the MERGE statement.

8) The output procedure may consist of any procedure needed to select, modify, or copy the records that are made available one at a time by the RETURN statement in merged order from the file referenced by file-name-1. The range includes all statements that are executed as the result of a transfer of control in the range of the output procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure. The range of the output procedure shall not cause the execution of any MERGE, RELEASE, or the file format of the SORT statement. (See 14.6.3, Explicit and implicit transfers of control.) If this rule is violated, the EC-SORT-MERGE-ACTIVE exception condition is set to exist and the results of the merge operation are undefined.

9) If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism after the last statement in the output procedure. When control passes to that return mechanism, the mechanism provides for termination of the merge, and then passes control to the next executable statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it selects the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

NOTE This return mechanism transfers control from the end of the output procedure and is not associated with the RETURN statement.

10) During the execution of the output procedure, no statement may be executed manipulating the file referenced by or accessing the record area associated with file-name-2 or file-name-3.

11) During the execution of any USE procedure implicitly invoked while executing the MERGE statement, no statement may be executed manipulating the file referenced by or accessing the record area associated with file-name-2, file-name-3, or file-name-4.

12) If the GIVING phrase is specified, all the merged records are written on each file referenced by file-name-4 as the implied output procedure for the MERGE statement. At the start of execution of the MERGE statement, the file referenced by file-name-4 shall not be in the open mode. If the file is in an open mode, the EC-SORT-MERGE-FILE-OPEN exception condition is set to exist and the execution of the MERGE statement terminates. For each of the files referenced by file-name-4, the execution of the MERGE statement causes the following actions to be taken:

a) The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT and SHARING WITH NO OTHER phrases had been executed. If a fatal exception condition exists as a result of this implicit OPEN statement and there is an applicable USE procedure that completes normally, processing for the file connector that caused the exception condition is bypassed. If a nonfatal exception condition exists as a result of this implicit OPEN statement and there is an applicable USE procedure that completes normally, the file connector that caused the exception condition is processed as if the exception did not exist.

b) The merged logical records are returned and written onto the file. Each record is written as if a WRITE statement without any optional phrases had been executed. If the file referenced by file-name-4 is described with variable-length records, the size of any record written to file-name-4 is the size of that record when it was read from file-name-1, regardless of the content of the data item referenced by the DEPENDING ON phrase of either a RECORD IS VARYING clause or an OCCURS clause specified in the file description entry for file-name-4. If an exception condition exists as a result of this implicit WRITE statement and there is an applicable USE procedure that completes normally, the MERGE continues execution, otherwise the MERGE statement is terminated.

   For a relative file, the relative key data item for the first record returned has the value 1; for the second record returned, the value 2; etc. After execution of the MERGE statement, the content of the relative key data item indicates the last record returned to the file.

c) The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

These implicit functions are performed such that any associated USE procedures are executed; however, the execution of such a USE procedure shall not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, file-name-4. On the first attempt to write beyond the externally defined boundaries of the file, any USE procedure associated with the file connector referenced by file-name-4 is executed; if control is returned from that USE procedure or if no such USE procedure is specified, the processing of the file is terminated as in general rule 12c above.

The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the sort-merge file description entry for file-name-1 is undefined upon completion of the MERGE statement for which the GIVING phrase is specified.

13) If the file referenced by file-name-4 contains only fixed-length records, any record in the file referenced by file-name-1 containing fewer character positions than that fixed-length is space filled on the right to that fixed length, beginning with the first character position after the last character in the record, when that record is returned to the file referenced by file-name-4, as follows:

a) If there is only one record description entry associated with the file referenced by file-name-4 and that record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.

b) If there are multiple record description entries associated with the file referenced by file-name-4 and the descriptions include a SELECT WHEN clause, the rules of the SELECT WHEN clause are applied to the record to select its description. When the record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.

c) Otherwise, the record is filled with alphanumeric space characters.

14) Additional rules affecting the execution of the MERGE statement are given in 12.4.4, File control entry, general rules 3 and 4.

14.9.24 MOVE statement

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

14.9.24.1 General format

Format 1 (simple):

```
MOVE { identifier-1 } TO { identifier-2 } ...
     { literal-1    }
```

Format 2 (corresponding):

```
MOVE { CORRESPONDING } identifier-3 TO identifier-4
     { CORR          }
```

14.9.24.2 Syntax rules

FORMAT 1

1) The class of identifier-1 or identifier-2 shall not be index, object, or pointer.

2) If identifier-2 references a strongly-typed group item, identifier-1 shall be specified and be described as a group item of the same type.

3) Literal-1 and the data item referenced by identifier-1 are sending operands.

4) Identifier-2 is a receiving operand.

5) The figurative constant SPACE shall not be moved to a numeric or numeric-edited item.

6) The figurative constant ZERO shall not be moved to an alphabetic data item.

7) Any figurative constant for which the associated character or characters are not boolean characters shall not be moved to a boolean data item.

8) If identifier-1 references a data item described with usage binary-char, binary-short, binary-long, or binary-double, identifier-2 shall reference a numeric or numeric-edited item.

9) If identifier-1 or identifier-2 references a variable-length group then these groups shall be compatible groups as specified in 8.5.1.9, Variable-length groups.

10) For all other cases not described in syntax rules 8 and 9, table 17, Validity of types of MOVE statements, specifies the validity of the move.

Table 17 — Validity of types of MOVE statements

Category of sending operand		Category of receiving operand					
		Alphabetic	**Alphanumeric -edited, Alphanumeric**	**Boolean**	**National, National-edited**	**Numeric, Numeric-edited**	**Type-name**
Alphabetic		Yes	Yes	No	Yes	No	No
Alphanumeric		Yes	Yes	Yes	Yes	Yes	No
Alphanumeric-edited		Yes	Yes	No	Yes	No	No
Boolean		No	Yes	Yes	Yes	No	No
National		No	No	Yes	Yes	Yes	No
National-edited		No	No	No	Yes	No	No
Numeric	Integer	No	Yes	No	Yes	Yes	No
	Noninteger	No	No	No	No	Yes	No
Numeric-edited		No	Yes	No	Yes	Yes	No
Type-name		Yes	Yes	Yes	Yes	Yes	Yes

FORMAT 2

11) The words CORR and CORRESPONDING are equivalent.

12) Identifier-3 and identifier-4 shall specify group data items and shall not be reference modified.

13) The corresponding data items within identifier-3 are sending operands. The corresponding data items within identifier-4 are receiving operands. The corresponding data items are determined according to the rules specified in 14.7.5, CORRESPONDING phrase.

14.9.24.3 General rules

FORMAT 1

1) Literal-1 or the content of the data item referenced by identifier-1 is moved to the data item referenced by each identifier-2 in the order specified.

   Item identification for identifier-2 is performed immediately before the data is moved to the respective data item. If identifier-2 is a zero-length item, the MOVE statement leaves identifier-2 unchanged.

   If identifier-1 is reference modified, subscripted, or is a function-identifier, the reference modifier, subscript, or function-identifier is evaluated only once, immediately before data is moved to the first of the receiving operands.

   The length of the data item referenced by identifier-1 is evaluated only once, immediately before the data is moved to the first of the receiving operands. If identifier-1 is a zero-length item, it is as if literal-1 were specified as a zero-lenght literal.

   The evaluation of the length of identifier-1 or identifier-2 may be affected by the DEPENDING ON phrase of the OCCURS clause.

   The result of the statement

```
MOVE a (b) TO b, c (b)
```

is equivalent to:

```
MOVE a (b) TO temp
MOVE temp TO b
MOVE temp to c (b)
```

where 'temp' is an intermediate result item provided by the implementor.

2) If literal-1 is not a boolean zero-length literal and the receiving operand is other than an any-length data item, literal-1 is treated as if it were the figurative constant SPACE. If literal-1 is a boolean zero-length literal and the receiving operand is other than an any-length data item, literal-1 is treated as if it were the figurative constant ZERO.

3) Any move in which the sending operand is either a literal or an elementary item and the receiving item is an elementary item is an elementary move. Bit group items and national group items are treated as elementary items in the MOVE statement.

   Any move that is not an elementary move, and does not reference a variable-length group, is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as specified in general rule 8 of the OCCURS clause and except as may be required by an implementation for subordinate items of class object or pointer.

4) De-editing takes place only when the sending operand is a numeric-edited data item and the receiving item is a numeric or a numeric-edited data item.

5) Any necessary conversion of data from one form of internal representation to another takes place during valid elementary moves, along with any editing specified for, or de-editing implied by, the receiving data item. When a national group item is referenced in a MOVE statement, no editing or de-editing takes place.

   NOTE Bit group items and national group items are treated as elementary items in the MOVE statement.

   Any necessary conversion from alphanumeric character to national character representation shall be performed, before any alignment, in accordance with a correspondence defined by the implementor. If no correspondence exists for any given alphanumeric character in the sending item, an implementor-defined substitution character is used as the corresponding national character in the receiving item and the EC-DATA-CONVERSION exception condition is set to exist.

   The following rules apply:

   a) When an alphanumeric, alphanumeric-edited, national, or national-edited data item is a receiving operand, alignment and any necessary space filling shall take place as defined in 14.6.8, Alignment of data within data items. If the sending operand is described as being signed numeric, the operational sign is not moved; if the operational sign occupies a separate character position, that character is not moved and the size of the sending operand is considered to be one less than its actual size. (See 13.18.51, SIGN clause.) If the usage of the sending operand is different from that of the receiving operand, conversion of the sending operand to the internal representation of the receiving operand takes place. If the sending operand is numeric and contains the picture symbol 'P', all digit positions specified with this symbol are considered to have the value zero and are counted in the size of the sending operand.

      If the sending item is of class boolean, its boolean value shall be moved.

      NOTE When the runtime coded character set is the UTF-16 format of the UCS, the COBOL system does not detect truncation that bisects the two halves of a surrogate pair.

b) When the sending operand and a receiving operand are identified as referencing the same data item, then:

   1. If the category of the operand is alphanumeric-edited or national-edited, the result of execution of the statement is undefined.

   2. If the operand is a variable-length data item, the result of execution of the statement is the same as if the content of the sending operand had been moved to a temporary fixed-length data item of the same class, category and length as the sending operand, and the content of that temporary data item were then moved to the receiving operand.

c) When a numeric item described with an IEEE floating-point usage is the receiving item in an elementary move, and a numeric item described with the same usage is the sending item, the information in the sending operand is transferred to the receiving item without change.

   NOTE This provision allows the preservation of the exact contents of sending data items containing subnormal finite numeric values, signaling and quiet NaN representations and infinity representations as well as normalized finite numeric values, into the receiving data item,

d) When a numeric item described with an IEEE binary floating-point usage is the receiving item in an elementary move, and a numeric item described with a different IEEE binary floating-point usage is the sending item, the following rules apply:

   1. If the content of the sending operand would evaluate to true in an infinity class condition and to true in a sign condition with the POSITIVE phrase, the content of the receiving operand is as if a SET CONTENT OF identifier-2 TO POSITIVE-INFINITY had been executed;

   2. If the content of the sending operand would evaluate to true in an infinity class condition and to true in a sign condition with the NEGATIVE phrase, the content of the receiving operand is as if a SET CONTENT OF identifier-2 TO NEGATIVE-INFINITY had been executed;

   3. If the content of the sending operand would evaluate to false in an infinity class condition and to false in a numeric class condition, the content of the receiving operand is as if a SET CONTENT OF identifier-2 TO NOT-A-NUMBER had been executed, with the additional provisions that the sign and signaling or quiet status of the sending operand (as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings) are also reflected in the receiving operand.

   4. If none of the above conditions are met, general rule 5g applies.

e) When a numeric item described with an IEEE decimal floating-point usage is the receiving item in an elementary move, and a numeric item described with a different IEEE floating-point usage is the sending item, the following rules apply:

   1. If the content of the sending operand would evaluate to true in an infinity class condition and to true in a sign condition with the POSITIVE phrase, the content of the receiving operand is as if a SET CONTENT OF identifier-2 TO POSITIVE-INFINITY had been executed;

   2. If the content of the sending operand would evaluate to true in an infinity class condition and to true in a sign condition with the NEGATIVE phrase, the content of the receiving operand is as if a SET CONTENT OF identifier-2 TO NEGATIVE-INFINITY had been executed;

   3. If the content of the sending operand would evaluate to false in an infinity class condition, and to true in a REPRESENTS-NOT-A-NUMBER class condition, the content of the receiving operand is as if a SET CONTENT OF identifier-2 TO NOT-A-NUMBER had been executed, with the additional provisions that the sign and signaling or quiet status of the sending operand (as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.5, Decimal interchange format encodings) are also reflected in the receiving operand;

4. If the content of the sending operand would evaluate to false in a REPRESENTS-NOT-A-NUMBER class condition, to false in an infinity class condition and to false in a numeric class condition, the EC-DATA-NOT-DECIMAL-ENCODING exception condition is set to exist, and the content of the receiving item is undefined;

5. If the content of the sending operand is of an IEEE floating-point usage, and is not in IEEE 754 Decimal128 decimal-encoded canonical form (as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.5, Decimal interchange format encodings, decimal encoding), the content is converted to a standard-decimal intermediate data item and that standard-decimal intermediate data item is used as the sending operand;

6. If none of the above conditions are met, general rule 5g applies.

f) If standard-binary arithmetic is in effect, the value in the sending operand is converted to a standard-binary intermediate data item, that standard-binary intermediate data item is used as the sending operand, and general rule 5g applies. If standard-decimal arithmetic is in effect, the value in the sending operand is converted to a standard-decimal intermediate data item, that standard-decimal intermediate data item is used as the sending operand, and general rule 5g applies.

g) When a numeric or numeric-edited item is the receiving item, and neither general rule 5c, general rule 5d, nor general rule 5e applies, alignment by decimal point and any necessary zero filling takes place as defined in 14.6.8, Alignment of data within data items or, if the LOCALE phrase of the PICTURE clause is specified, as defined in 13.18.39.4, Editing rules. When the sending operand is numeric-edited, de-editing is implied to establish the operand's unedited numeric value, which may be signed; then the unedited numeric value is moved to the receiving field. Restrictions on the content of a numeric-edited sending operand are given in 14.6.12.2, Incompatible data. When the category of the sending operand is other than numeric-edited and the content of the sending operand would result in a false value in a numeric class condition, the EC-DATA-INCOMPATIBLE exception condition is set to exist and the results of the execution of the MOVE statement are undefined.

1. When the sending operand is a floating-point item, and the receiving operand is a fixed-point item, the content of the sending operand is treated as if it had been converted to a fixed-point operand of the same value.

2. When a signed numeric item is the receiving item, the sign of the sending operand is placed in the receiving item. (See 13.18.51, SIGN clause.) Conversion of the representation of the sign takes place as necessary. If the sending operand is unsigned, a positive sign is generated for the receiving item.

3. When an unsigned numeric item is the receiving item, the absolute value of the sending operand is moved and no operational sign is generated for the receiving item.

4. When the sending operand is described as alphanumeric or national, the sending operand is treated as if it were an unsigned integer of category numeric with the following characteristics:

   a. If the sending operand is a data item, the number of digits is the number of character positions in the sending data item unless the number of character positions is greater than 31, in which case the rightmost 31 character positions are used.

   b. If the sending operand is a figurative constant, the number of digits is the same as the number of digits in the receiving operand and the figurative constant is replicated in this item, from left to right, as described in the rules for figurative constants. If the receiving item is not an integer, the number of digits includes both those to the right and the left of the decimal point.

   c. If the sending operand is an alphanumeric or national literal, the number of digits is the same as the number of characters in the literal. If the number of characters exceeds 31, the size of the sending operand is 31 digits and only the rightmost 31 alphanumeric characters in the literal are used.

h) When a receiving item is described as alphabetic, justification and any necessary space filling takes place as defined in 14.6.8, Alignment of data within data items.

i) When a boolean data item is a receiving item, the boolean value of the sending item is moved, with conversion of data representation if necessary, and any necessary alignment and zero filling takes place as defined in 14.6.8, Alignment of data within data items. If the category of the sending operand is not boolean, the sending operand is treated as if it were a boolean data item with the same usage as the actual sending operand and with the same number of character positions as the actual sending operand. If the content of the sending operand would result in a false value in a boolean class condition, the EC-DATA-INCOMPATIBLE exception condition is set to exist and the results of the execution of the MOVE statement are undefined.

6) Alphanumeric, boolean, national, and numeric literals belong to the categories alphanumeric, boolean, national, and numeric, respectively. The category of figurative constants when used in the MOVE statement depends on the category of the receiving operand as shown in table 18, Category of figurative constants used in the MOVE statement.

NOTE MOVE of the figurative constant QUOTE or QUOTES to a numeric data item is an obsolete feature and is to be removed from the next edition of standard COBOL. MOVE of figurative constants that are not numeric, other than QUOTE or QUOTES, to a numeric item is an archaic feature of standard COBOL and its use should be avoided.

Table 18 — Category of figurative constants used in the MOVE statement

Figurative constant	Category of receiving operand	Category of figurative constant
ALL literal, where literal is: Alphanumeric Boolean National	 — — —	 Alphanumeric Boolean National
ALL symbolic character, where symbolic character is: Alphanumeric National	 — —	 Alphanumeric National
HIGH-VALUE, HIGH-VALUES; LOW-VALUE, LOW-VALUES; and QUOTE, QUOTES	Alphabetic Alphanumeric Alphanumeric-edited National National-edited Numeric Numeric-edited — if usage is display — if usage is national	Alphanumeric Alphanumeric Alphanumeric National National Alphanumeric Alphanumeric National
SPACE, SPACES	Alphabetic Alphanumeric Alphanumeric-edited National National-edited	Alphanumeric Alphanumeric Alphanumeric National National
ZERO, ZEROS, ZEROES	Alphanumeric Alphanumeric-edited Boolean National National-edited Numeric Numeric-edited	Alphanumeric Alphanumeric Boolean National National Numeric Numeric
— indicates the figurative constant category does not depend on the category of the receiving operand		

7) If the sending or receiving item is an any-length elementary item, the current content of the any-length elementary item is moved or changed as specified in 8.5.1.7.3, Operations on any-length elementary items.

8) If either the sending or the receiving item, or both, is a variable-length group, the following rules apply:

   a) If the groups are of equal length:

      The content of each character position that is not occupied by a corresponding table is moved to the corresponding character position in the receiving group.

      Where two tables correspond, as specified in 8.5.1.9.1, Positional correspondence, the table in the sending group is moved to the corresponding table in the receiving group, as specified in 8.5.1.6.3.5.1, Moving a table.

   b) If the groups are of unequal length:

      If the sending group is longer than the receiving group, the character positions that occupy the excess part are ignored by the operation.

If the sending group is shorter than the receiving group, each location that occupies the excess part is space filled, according to the following recursive procedure:

1. If the data item to be space-filled is an any-length elementary item, the length of the receiving operand is set to zero.

2. If the data item to be space filled is a dynamic-capacity table, it is space filled according to 8.5.1.6.3.5.3, Space filling a dynamic table.

3. All other character positions are filled with space characters.

The common part of the variable-length groups is now moved according to the rules for variable-length groups of equal length, as defined in general rule 8a above.

NOTE If the receiving table is an occurs-depending table, it is the programmer's responsibility to store an appropriate value in the data item associated with the DEPENDING phrase independently of the MOVE statement, because this data item is not changed by the operation.

9) Additional rules and explanations relative to this statement are given in 14.6.9, Overlapping operands.

FORMAT 2

10) Data items within identifier-3 are selected to be moved to selected data items within identifier-4 according to the rules specified in 14.7.5, CORRESPONDING phrase. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements except that if subscripting is specified for identifier-3 or identifier-4, the subscript value used is that resulting from the evaluation of the subscript at the start of the execution of the statement.

NOTE For purposes of MOVE CORRESPONDING, bit group items and national group items are processed as group items, rather than as elementary items.

14.9.25 MULTIPLY statement

The MULTIPLY statement causes numeric data items to be multiplied and sets the values of data items equal to the results.

14.9.25.1 General format

Format 1 (by):

```
MULTIPLY { identifier-1 } BY { identifier-2 [ rounded-phrase ] } ...
         { literal-1    }
  [ ON SIZE ERROR imperative-statement-1 ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-MULTIPLY ]
```

Format 2 (by-giving):

```
MULTIPLY { identifier-1 } BY { identifier-2 }
         { literal-1    }    { literal-2    }
  GIVING { identifier-3 [ rounded-phrase ] } ...
  [ ON SIZE ERROR imperative-statement-1 ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-MULTIPLY ]
```

where rounded-phrase is described in 14.7.3, ROUNDED phrase.

14.9.25.2 Syntax rules

1) Identifier-1 and identifier-2 shall reference a data item of category numeric.

2) Identifier-3 shall reference a data item of category numeric or numeric-edited.

3) Literal-1 and literal-2 shall be numeric literals.

4) When native arithmetic is in effect, the composite of operands described in 14.7.6, Arithmetic statements, is determined by using all of the operands in the statement.

14.9.25.3 General rules

1) When format 1 is used and native arithmetic is in effect, the initial evaluation consists of determining the multiplier, which is literal-1 or the value of the data item referenced by identifier-1. The multiplicand is the value of the data item referenced by identifier-2. The product of the multiplier and the multiplicand is stored as the new value of the data item referenced by identifier-2.

2) When format 2 is used and native arithmetic is in effect, the initial evaluation consists of determining the multiplier, which is literal-1 or the value of the data item referenced by identifier-1; determining the multiplicand, which is literal-2 or the value of the data item referenced by identifier-2; and forming the product of the multiplier and the multiplicand. The product is stored as the new value of each data item referenced by identifier-3.

3) When format 1 or 2 is used and standard, standard-decimal, or standard-binary arithmetic is in effect, the product equals the result of the arithmetic expression

    (multiplier * multiplicand)

    where the values for multiplier and multiplicand are as defined in general rules 1 and 2 for the respective formats.

4) Additional rules and explanations relative to this statement are given in 14.6.12.2, Incompatible data; 14.7.3, ROUNDED phrase; 14.7.4, SIZE ERROR phrase and size error condition; and 14.7.6, Arithmetic statements.

14.9.26 OPEN statement

The OPEN statement initiates the processing of files.

14.9.26.1 General format

```
OPEN { { INPUT  }                                                                  }
     { { OUTPUT } [ sharing-phrase ] [ retry-phrase ] { file-name-1 [ WITH NO REWIND ] } ... } ...
     { { I-O    }                                                                  }
     { { EXTEND }                                                                  }
```

where sharing-phrase is:

```
SHARING WITH { ALL OTHER  }
             { NO OTHER   }
             { READ ONLY  }
```

where retry-phrase is described in 14.7.8, RETRY phrase

14.9.26.2 Syntax rules

1) The OPEN statement for a report file shall not contain the INPUT phrase or the I-O phrase.

2) The EXTEND phrase shall be specified only if the access mode of the file connector referenced by file-name-1 is sequential and the LINAGE clause is not specified in the file description entry for file-name-1.

3) The files referenced in the OPEN statement need not all have the same organization or access.

4) The NO REWIND phrase may be specified only for sequential files.

5) The NO REWIND phrase may be specified only when the INPUT or OUTPUT phrase is specified.

6) If the SHARING phrase is omitted from the OPEN statement and the ALL phrase is specified in the SHARING clause of the file control entry for file-name-1 or if the ALL phrase is specified on the OPEN statement, the LOCK MODE clause shall be specified in the file control entry for file-name-1.

7) The I-O phase shall not be specified if the FORMAT clause is specified in the file description entry for file-name-1.

14.9.26.3 General rules

1) The file connector referenced by file-name-1 shall not be open. If it is open, the execution of the OPEN statement is unsuccessful and the I-O status associated with file-name-1 is set to '41'.

2) The successful execution of an OPEN statement associates the file connector referenced by file-name-1 with a physical file if the physical file is available, and sets the open mode of the file connector to input, output, I-O, or extend, depending on the keywords INPUT, OUTPUT, I-O or EXTEND specified in the OPEN statement. The open mode determines the input-output statements that are allowed to reference the file connector as shown in table 21, Permissible I-O statements by access mode and open mode.

   A physical file is available if it is physically present and is recognized by the operating environment. Table 19, Opening available and unavailable files (file not currently open), shows the results of opening available and unavailable physical files that are not currently open by another file connector. Table 20, Opening available

shared files that are currently open by another file connector, shows the results of opening available physical files that are currently open by another file connector, including those implicitly opened by the SORT and MERGE statements.

Table 19 — Opening available and unavailable files (file not currently open)

Open mode	File is available	File is unavailable
INPUT	Normal open	Open is unsuccessful
INPUT (optional file)	Normal open	Normal open; the first read causes the at end condition or invalid key condition
I-O	Normal open	Open is unsuccessful
I-O (optional file)	Normal open	Open causes the file to be created
OUTPUT	Normal open; the file contains no records	Open causes the file to be created
EXTEND	Normal open	Open is unsuccessful
EXTEND (optional file)	Normal open	Open causes the file to be created

Table 20 — Opening available shared files that are currently open by another file connector

Open request		Most restrictive existing sharing mode and open mode				
		sharing with no other	sharing with read only		sharing with all other	
		extend I-O input output	extend I-O output	input	extend I-O output	input
SHARING WITH NO OTHER	EXTEND I-O INPUT OUTPUT	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open
SHARING WITH READ ONLY	EXTEND I-O	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open	Normal open
	INPUT	Unsuccessful open	Unsuccessful open	Normal open	Unsuccessful open	Normal open
	OUTPUT	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open
SHARING WITH ALL OTHER	EXTEND I-O	Unsuccessful open	Unsuccessful open	Unsuccessful open	Normal open	Normal open
	INPUT	Unsuccessful open	Normal open	Normal open	Normal open	Normal open
	OUTPUT	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open

3) The successful execution of an OPEN statement makes the associated record area available to the runtime element. If the file connector associated with file-name-1 is an external file connector, there is only one record area associated with the file connector for the run unit.

4) When a file connector is not open, no statement shall be executed that references the associated file-name, either explicitly or implicitly, except for a MERGE or SORT statement with the USING or GIVING phrase, or an OPEN statement.

5) The OPEN statement for a report file connector shall be executed before the execution of an INITIATE statement that references a report-name that is associated with file-name-1.

6) For a given file connector, an OPEN statement shall be successfully executed prior to the execution of any other permissible input-output statement. In table 21, Permissible I-O statements by access mode and open mode, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the open mode given at the top of the column.

Table 21 — Permissible I-O statements by access mode and open mode

Access mode	Statement	Open mode			
		Input	Output	I-O	Extend
Sequential	READ	X		X	
	WRITE		X		X
	REWRITE			X	
	START	X		X	
Sequential (relative and indexed files only)	DELETE			X	
Random	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
Dynamic	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

7) Execution of the OPEN statement does not obtain or release the first record.

8) During the execution of the OPEN statement when the file connector is matched with the physical file and the physical file exists, the attributes of the file connector as specified in the file control paragraph and the file description entry are compared with the fixed file attributes of the physical file. If the attributes don't match, a file attribute conflict condition occurs, the execution of the OPEN statement is unsuccessful, and the I-O status associated with file-name-1 is set to '39'. The implementor defines which of the fixed-file attributes are validated during the execution of the OPEN statement. The validation of fixed-file attributes may vary depending on the organization or storage medium of the file. (See 9.1.6, Fixed file attributes.)

9) The NO REWIND phrase will be ignored if it does not apply to the storage medium on which the physical file resides. If the NO REWIND phrase is ignored, the OPEN statement is successful and the I-O status associated with file-name-1 is set to '07'.

10) If the storage medium for the physical file permits rewinding, the following rules apply:

    a) When neither the EXTEND, nor the NO REWIND phrase is specified, execution of the OPEN statement causes the physical file to be positioned at its beginning.

    b) When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the physical file to be repositioned; the physical file shall be already positioned at its beginning prior to execution of the OPEN statement.

11) If the physical file is not present, and the INPUT phrase is specified in the OPEN statement, and the OPTIONAL clause is specified in the file control entry for file-name-1, the file position indicator in the file connector referenced by file-name-1 is set to indicate that an optional input file is not present.

12) When the organization of the file referenced by file-name-1 is sequential or relative and the INPUT or I-O phrase is specified in the OPEN statement, the file position indicator for that file connector is set to 1. When the organization is indexed, the file position indicator is set to the characters that have the lowest ordinal position in the collating sequence associated with the file, and the prime record key is established as the key of reference.

13) When the EXTEND phrase is specified, the OPEN statement positions the file immediately after the last logical record for that file. The last logical record for a sequential file is the last record written in the file. The last logical record for a relative file is the currently existing record with the highest relative record number. The last logical record for an indexed file is the currently existing record with the highest prime key value.

14) If the I-O phrase is specified, the physical file shall support the input and output statements that are permitted for the organization of that file when opened in the I-O mode. If the physical file does not support those statements, the I-O status value for file-name-1 is set to '37' and the execution of the OPEN statement is unsuccessful. The successful execution of an OPEN statement with the I-O phrase sets the open mode of file connector referenced by file-name-1 to open in the I-O mode.

15) If the physical file is not present, and the EXTEND or I-O phrase is specified in the OPEN statement, and the OPTIONAL clause is specified in the file control entry for file-name-1, the OPEN statement creates the file. This creation takes place as if the following statements were executed in the order shown:

    ```
    OPEN OUTPUT file-name.
    CLOSE file-name.
    ```

    These statements are followed by execution of the OPEN statement specified in the source element.

16) If the OUTPUT phrase is specified, the successful execution of the OPEN statement creates the physical file. After the creation of the physical file, the file contains no records. If physical pages have meaning for the physical file, the positioning of the output medium with respect to physical page boundaries is implementor-defined following the successful execution of the OPEN statement, whether or not the LINAGE clause is specified in the file description entry of file-name-1.

17) Upon successful execution of the OPEN statement, the current volume pointer is set:

    a) To point to the first or only reel/unit in the physical file if INPUT or I-O is specified.

    b) To point to the reel/unit containing the last record in the physical file if EXTEND is specified.

    c) To point to the newly created reel/unit in the physical file for an unavailable file if EXTEND, I-O, or OUTPUT is specified.

18) The execution of the OPEN statement causes the value of the I-O status associated with file-name-1 to be updated to one of the values in 9.1.12, I-O status.

19) If more than one file-name is specified in an OPEN statement, the result of executing this OPEN statement is the same as if a separate OPEN statement had been written for each file-name in the same order as specified in the OPEN statement. These separate OPEN statements would each have the same open mode specification, the sharing-phrase, retry-phrase, and REWIND phrase as specified in the OPEN statement. If an implicit OPEN statement results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, processing resumes at the next implicit OPEN statement, if any.

20) The SHARING phrase is effective only for files that are shareable.

21) The SHARING phrase specifies the level of sharing permitted for the physical file associated with file-name-1 and specifies the operations that may be performed on the physical file through other file connectors sharing the physical file, as indicated in 9.1.14, Sharing mode.

22) The SHARING phrase overrides any SHARING clause in the file control entry of file-name-1. If there is no SHARING phrase on the OPEN statement, then file sharing is completely specified in the file control entry. If neither a SHARING phrase on the OPEN statement nor a SHARING clause in the file control entry is specified, the implementor shall define the sharing mode that is established for each file connector.

23) The RETRY phrase is used to control the behavior of an OPEN statement when the open mode or sharing mode requested conflicts with those of other file connectors that are currently associated with the physical file. The I-O status is set in accordance with the rules in 14.7.8, RETRY phrase.

24) If the file connector referenced by file-name-1 is locked by a previously-executed CLOSE statement with the LOCK phrase, the execution of the OPEN statement is unsuccessful and the I-O status associated with that file connector is set to '38'.

25) If the execution of the OPEN statement is unsuccessful, the physical file is not affected and the following actions take place in the following order:

    a) A value is placed in the I-O status associated with file-name to indicate the condition that caused the OPEN statement to be unsuccessful.

    b) The level-3 EC-I-O exception condition associated with the I-O status value is set to exist.

    c) Any applicable USE FOR EXCEPTION or USE AFTER EXCEPTION procedure is executed as specified for the rules for the USE statement.

26) Additional rules affecting the execution of the OPEN statement are given in 12.4.4, File control entry, general rules 3 and 4.

14.9.27 PERFORM statement

The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete. The PERFORM statement is also used to control execution of one or more imperative statements that are within the scope of that PERFORM statement.

14.9.27.1 General format

Format 1 (out-of-line):

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ] [ times-phrase ]
                         [ { THRU    }                  ] [ until-phrase ]
                                                          [ varying-phrase ]
```

Format 2 (inline):

```
PERFORM [ times-phrase ]
        [ until-phrase ] imperative-statement-1 END-PERFORM
        [ varying-phrase ]
```

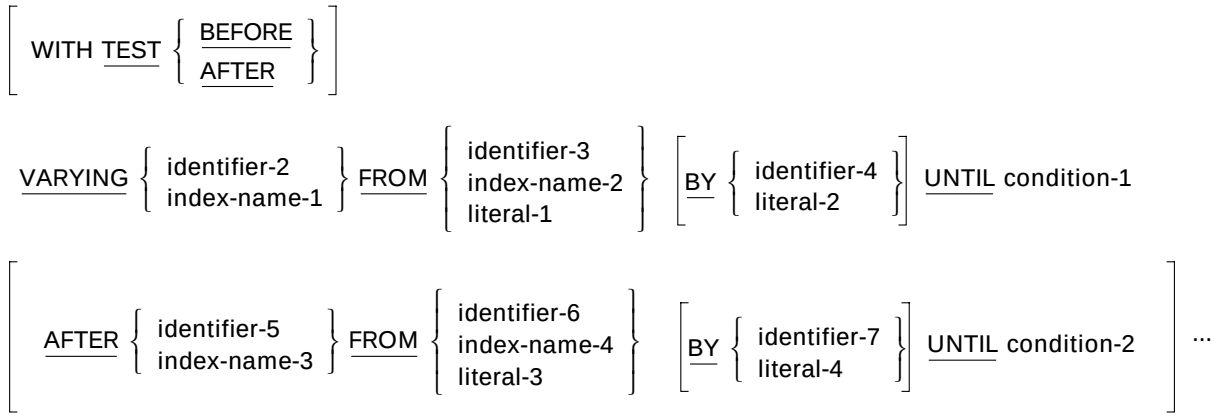
where times-phrase is:

```
{ identifier-1 } TIMES
{ integer-1    }
```

where until-phrase is:

```
[ WITH TEST { BEFORE } ] UNTIL condition-1
            { AFTER  }
```

where varying-phrase is:



14.9.27.2 Syntax rules

1) If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.

2) Each identifier shall reference a numeric elementary item described in the data division. Identifier-1 shall be an integer.

3) Each literal shall be numeric.

4) The words THROUGH and THRU are equivalent.

5) If an index-name is specified in the VARYING or AFTER phrase, then:

   a) The identifier in the associated FROM and BY phrases shall reference an integer data item.

   b) The literal in the associated FROM phrase shall be a positive integer.

   c) The literal in the associated BY phrase shall be a nonzero integer.

6) If an index-name is specified in the FROM phrase, then:

   a) The identifier in the associated VARYING or AFTER phrase shall reference an integer data item.

   b) The identifier in the associated BY phrase shall reference an integer data item.

   c) The literal in the associated BY phrase shall be an integer.

7) The literal in the BY phrase shall not be zero.

8) Condition-1, condition-2, ... , may be any conditional expression. (See 8.8.4, Conditional expressions.)

9) When procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the declaratives portion of the procedure division, both shall be procedure-names in the same declarative section.

10) At least six AFTER phrases shall be permitted in varying-phrase.

11) Procedure-name-1 shall be the name of either a paragraph or a section in the same source element as that in which the PERFORM statement is specified.

12) Procedure-name-2 shall be the name of either a paragraph or a section in the same source element as that in which the PERFORM statement is specified.

14.9.27.3 General rules

1) If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, at the time the data item referenced by the identifier is used to initialize the index associated with the index-name, the data item shall have a positive value. If the data item does not have a positive value, the EC-RANGE-PERFORM-VARYING exception condition is set to exist.

2) An inline PERFORM statement and an out-of-line PERFORM statement function identically according to the following rules. For an out-of-line PERFORM statement, the specified set of statements consists of all statements beginning with the first statement of procedure-name-1 and ending with the last statement of procedure-name-2, or, if procedure-name-2 is not specified, the last statement of procedure-name-1. For an inline PERFORM statement, the specified set of statements consists of all statements contained within the PERFORM statement.

3) When the PERFORM statement is executed, control is transferred to the first statement of the specified set of statements except as indicated in general rules 7, 8, and 9. For those cases where a transfer of control to the specified set of statements does take place, an implicit transfer of control to the end of the PERFORM statement is established as follows:

   a) If procedure-name-2 is not specified, the return mechanism is after the last statement of procedure-name-1.

   b) If procedure-name-2 is specified, the return mechanism is after the last statement of procedure-name-2.

4) There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2.

   NOTE Statements such as the GO TO statement, the PERFORM statement, and the procedure format of the EXIT statement may occur in the flow of execution of the specified set of statements, however the flow of execution should eventually pass to the end of procedure-name-2.

5) If control passes to the specified set of statements by means other than a PERFORM statement, control will pass through the last statement of the set to the next executable statement as if no PERFORM statement referenced the set.

6) A PERFORM statement without times-phrase, until-phrase, or varying-phrase is the basic PERFORM statement. The specified set of statements referenced by this type of PERFORM statement is executed once and then control passes to the end of the PERFORM statement.

7) If times-phrase is specified, the specified set of statements is performed the number of times specified by integer-1 or by the value of the data item referenced by identifier-1 at the start of the execution of the PERFORM statement. If at the start of the execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is negative, control passes to the end of the PERFORM statement. Following the execution of the specified set of statements the specified number of times, control is transferred to the end of the PERFORM statement.

   NOTE During execution of the PERFORM statement, a change to the contents of identifier-1 does not alter the number of times the specified set of statements is performed.

8) If until-phrase is specified, the specified set of statements is performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the end of the PERFORM statement. If the condition is true when the PERFORM statement is entered, and the TEST BEFORE phrase is specified or implied, no transfer to the specified set of statements takes place, and control is passed to the end of the PERFORM statement. If the TEST AFTER phrase is specified, the PERFORM statement functions as if the TEST BEFORE phrase were specified except that the condition is tested after the specified set of statements has been

executed. Item identification associated with the operands specified in condition-1 is done each time the condition is tested.

9) If varying-phrase is specified, the execution of the PERFORM statement augments the data items referenced by one or more identifiers or the indexes referenced by one or more index-names in an orderly fashion. In the following rules, the data items referenced by identifier-2 and identifier-5 and the indexes referenced by index-name-1 and index-name-3 are referred to as the induction variables. The content of the data item referenced by the identifier, the occurrence number corresponding to the value of the index referenced by the index-name, or the value of the literal referenced in the FROM phrase is referred to as the initialization value. The content of the data item referenced by the identifier or the value of the literal in a BY phrase is referred to as the augment value. For any BY phrase that is omitted, the augment value is 1. Item identification for identifier-2, identifier-5, index-name-1, or index-name-3 is done each time the content of the data item referenced by the identifier or the value of the index referenced by the index-name is set or augmented. Item identification for identifier-3, identifier-4, identifier-6, identifier-7, index-name-2, and index-name-4 is done each time the content of the data item referenced by the identifier or the index referenced by the index-name is used in a setting or augmenting operation. Item identification associated with the operands specified in condition-1 or condition-2 is done each time the condition is tested.

   NOTE If an augment value is less than 0, the induction variable is actually decremented by the absolute value of the augment value.

   The sequence of operation of the PERFORM statement is as follows:

   a) All induction variables are set to their associated initialization values in the left-to-right order in which the induction variables are specified.

   b) If the TEST AFTER phrase is specified, and there is no AFTER phrase, the specified set of statements is executed once and condition-1 is tested. If the condition is false, the induction variable is incremented by the augment value, and the specified set of statements is executed again. The cycle continues until condition-1 is tested and found to be true, at which point control is transferred to the end of the PERFORM statement. At that point, the induction variable contains the value it contained at the completion of the execution of the specified set of statements.

   c) If the TEST AFTER phrase is specified, and there is one or more AFTER phrase, the following occurs:

      1. The specified set of statements is executed.

      2. The rightmost condition-2 is then evaluated.

      3. If the rightmost condition-2 is false, the associated induction variable is incremented by the associated augment value, and execution proceeds with step a.

      4. If the last condition evaluated is true, the condition to its left is evaluated. This is repeated until either a false condition is found or the last condition evaluated is condition-1 and condition-1 is true. If a false condition is found, the induction variable associated with that condition is incremented by the associated augment value, all induction variables to the right of the false condition are set to their initialization values, and execution proceeds with step a. If no condition is found to be false, control is transferred to the end of the PERFORM statement.

         NOTE After successful execution of the PERFORM statement, all induction variables contain the values they had at the completion of the last execution of the specified set of statements.

   d) If the TEST AFTER phrase is not specified and there is no AFTER phrase, condition-1 is evaluated, and if it is true, control is transferred to the end of the PERFORM statement. If it is false, the specified set of statements is executed. Then, the induction variable is incremented by the augment value, and condition-1 is evaluated again. When control is passed to the end of the PERFORM statement, the induction variable contains the value it contained when condition-1 was evaluated.

   e) If the TEST AFTER phrase is not specified, and there is one or more AFTER phrase, the following occurs:

1. Condition-1 is evaluated.

   If condition-1 is true, control is transferred to the end of the PERFORM statement; otherwise, the condition-2 immediately to the right becomes the current condition.

2. The current condition is evaluated.

   If the current condition is true:

   a. the induction variable associated with the current condition is set to its initialization value

   b. the condition to the left of the current condition becomes the current condition

   c. the induction variable associated with the new current condition is incremented by its associated augment value

   d. if the current condition is condition-1 execution proceeds to step a, else execution proceeds to the beginning of step b.

   otherwise:

   a. if there is another AFTER phrase to the right of the current condition,

      —the condition associated with that AFTER phrase becomes the current condition

      —execution proceeds to the beginning of step b;

   b. otherwise

      —the specified set of statements is executed

      —the induction variable associated with the current condition is incremented by the augment value

      —execution proceeds to the beginning of step b.

   NOTE After successful execution of the PERFORM statement, all induction variables contain the values they had at the completion of the last evaluation of condition-1. With the exception of the induction variable associated with condition-1, these values are the same as they were at the last execution of the specified set of statements, or are their associated initialization values if no statements were executed. If no statements were executed, the induction variable associated with condition-1 contains its associated initialization value; otherwise, the induction variable associated with condition-1 contains the value it contained after the last execution of the specified set of statements, incremented by the augment value.

   During the execution of the specified set of statements associated with the PERFORM statement, all changes to the induction variable, the variables associated with the augment value, and the variables associated with the initialization value have immediate effect and all subsequent references to the associated data items use the updated contents.

10) The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the implicit transfer of control to the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control in the range of the PERFORM statement, except for statements executed as the result of a transfer of control by an EXIT FUNCTION, EXIT METHOD, EXIT PROGRAM, or GOBACK statement specified in the same instance of the same source element as the PERFORM statement. Declarative procedures that are executed as a result of the execution of statements in the range of a PERFORM statement are included in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source element.

11) The results of executing the following sequence of PERFORM statements are undefined and no exception condition is set to exist when the sequence is executed:

    a) a PERFORM statement is executed and has not yet terminated, then

    b) within the range of that PERFORM statement another PERFORM statement is executed, then

    c) the execution of the second PERFORM statement passes through the exit of the first PERFORM statement.

    NOTE Because this is undefined, the user should avoid such an execution sequence. On some implementations it causes stack overflows, on some it causes returns to unlikely places, and on other implementations other actions can occur. Therefore, the results are unpredictable and are unlikely to be portable.

14.9.28 RAISE statement

The RAISE statement causes a specified exception condition to be raised.

14.9.28.1 General Format

RAISE { EXCEPTION exception-name-1 | identifier-1 }

14.9.28.2 Syntax Rules

1) Exception-name-1 shall be a level-3 exception-name as specified in 14.6.12.1, Exception conditions.

2) Identifier-1 shall be an object reference; the predefined object references NULL and SUPER shall not be specified.

3) Identifier-1 is a sending operand.

14.9.28.3 General Rules

1) If exception-name-1 is specified, the associated exception condition is raised, and EXCEPTION-OBJECT is set to null. If there is no applicable declarative and the exception condition is nonfatal, processing continues with the statement following the RAISE statement.

2) If identifier-1 is specified, EXCEPTION-OBJECT is set to reference the object referenced by identifier-1. If there is no applicable declarative, processing continues with the statement following the RAISE statement.

14.9.29 READ statement

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a mass storage file.

14.9.29.1 General format

Format 1 (sequential):

```
READ file-name-1 { NEXT     } RECORD [ INTO identifier-1 ]
                 { PREVIOUS }

  [ ADVANCING ON LOCK ]
  [ IGNORING LOCK     ]
  [ retry-phrase      ]

  [ WITH LOCK    ]
  [ WITH NO LOCK ]

  [ AT END imperative-statement-1     ]
  [ NOT AT END imperative-statement-2 ]
[ END-READ ]
```

Format 2 (random):

```
READ file-name-1 RECORD [ INTO identifier-1 ]

  [ IGNORING LOCK ]
  [ retry-phrase  ]

  [ WITH LOCK    ]
  [ WITH NO LOCK ]

  [ KEY IS { data-name-1     } ]
           { record-key-name-1 }

  [ INVALID KEY imperative-statement-3     ]
  [ NOT INVALID KEY imperative-statement-4 ]
[ END-READ ]
```

where retry-phrase is described in 14.7.8, RETRY phrase

14.9.29.2 Syntax rules

ALL FORMATS

1) The INTO phrase may be specified in a READ statement:

   a) If no record description entry or only one record description is subordinate to the file description entry, or

   b) If the data item referenced by identifier-1 and all record-names associated with file-name-1 describe an alphanumeric group item or an elementary item of category alphanumeric or category national.

2) If identifier-1 is a strongly-typed group item, there shall be at most one record area subordinate to the FD for file-name-1. This record area, if specified, shall be a strongly-typed group item of the same type as identifier-1.

3) The LOCK phrase shall not be specified in the same READ statement as the IGNORING LOCK phrase.

4) If automatic locking has been specified for file-name-1, none of the phrases IGNORING LOCK, WITH LOCK, or WITH NO LOCK shall be specified.

FORMAT 1

5) None of the phrases ADVANCING, AT END, NEXT, NOT AT END, or PREVIOUS shall be specified if ACCESS MODE RANDOM is specified in the file control entry for file-name-1.

6) If neither the NEXT phrase nor the PREVIOUS phrase is specified and ACCESS MODE SEQUENTIAL is specified in the file control entry for file-name-1, the NEXT phrase is implied.

7) If neither the NEXT phrase nor the PREVIOUS phrase is specified and ACCESS MODE DYNAMIC is specified in the file control entry for file-name-1, the NEXT phrase is implied if any of the following phrases is specified: ADVANCING, AT END, or NOT AT END.

FORMAT 2

8) The KEY phrase may be specified only if ORGANIZATION IS INDEXED is specified in the file control entry for file-name-1.

9) Data-name-1 or record-key-name-1 shall be specified in the RECORD KEY clause or an ALTERNATE RECORD KEY clause associated with file-name-1.

10) Data-name-1 or record-key-name-1 may be qualified.

14.9.29.3 General rules

ALL FORMATS

1) The open mode of the file connector referenced by file-name-1 shall be input or I-O. If it is any other value, the execution of the READ statement is unsuccessful and the I-O status value for file-name-1 is set to '47'.

2) The execution of the READ statement causes the value of the I-O status in the file connector referenced by file-name-1 to be updated as indicated in 9.1.12, I-O status.

3) When the logical records of a file are described with more than one record description, the contents of any data item, where any part of that data item lies beyond the range of the current record, are undefined at the completion of the execution of the READ statement.

4) If the execution of a READ statement with the INTO phrase is successful, the result is equivalentt to the application of the following rules in the order specified:

   a) The same READ statement without the INTO phrase is executed.

   b) The current record is moved from the record area to the area specified by identifier-1 according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is determined by rules specified in the RECORD clause. If the file description entry contains a RECORD IS VARYING clause, the implied move is an alphanumeric group move. Item identification of the data item referenced by identifier-1 is done after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data item referenced by identifier-1.

      NOTE 14.6.9, Overlapping operands, and 14.9.24, MOVE statement, general rules, apply to any cases in which the storage area identified by identifier-1 and the record area associated with file-name-1 share any part of their

storage areas. The result of execution of the READ statement is undefined if the result of execution of the implicit MOVE statement described in general rule 4b is undefined.

5) If the execution of a READ statement with the INTO phrase is unsuccessful, the content of the data item referenced by identifier-1 is unchanged and item identification of the data item referenced by identifier-1 is not done.

6) The execution of a READ statement with the INTO phrase when there are no record description entries subordinate to the file description entry proceeds as though there were one record description entry describing an alphanumeric group item of the maximum size established by the RECORD clause.

7) Whether record locking is in effect is determined by the rules specified in 12.4.4.8, LOCK MODE clause.

8) If record locking is enabled for the file connector referenced by file-name-1 and the record identified for access by the general rules for the READ statement is locked by that file connector, the record lock is ignored and the READ operation proceeds as if the record were not locked.

9) If record locking is enabled for the file connector referenced by file-name-1 and the record identified for access is locked by another file connector, the result of the operation depends on the presence or absence of the RETRY phrase. If the RETRY phrase is specified, additional attempts may be made to read the record as specified in the rules in 14.7.8, RETRY phrase. If the RETRY phrase is not specified or the record is not successfully accessed as specified by the RETRY phrase, the record operation conflict condition exists. The I-O status is set in accordance with the rules for the RETRY phrase.

10) If the record operation conflict condition exists as a result of the READ statement:

    a) The file position indicator is unchanged.

    b) A value is placed into the I-O status associated with file-name-1 to indicate the record operation conflict condition.

    c) The content of the associated record area is undefined.

    d) The key of reference for indexed files is unchanged.

    e) The READ statement is unsuccessful.

11) If record locks are in effect, the following actions take place:

    a) If single record locking is specified for the file connector associated with file-name-1, any prior record lock associated with that file connector is released by the execution of the READ statement.

    b) If multiple record locking is specified for the file connector associated with file-name-1, no record locks are released, except when the NO LOCK phrase is specified and the record accessed was already locked by that file connector. In this case, that record lock is released at the completion of the successful execution of the READ statement.

    c) If the lock mode is automatic, the record lock associated with a successfully accessed record is set.

    d) If lock mode is manual, the record lock associated with a successfully accessed record is set only if the LOCK phrase is specified on the READ statement.

12) If the IGNORING LOCK phrase is specified on the READ statement, the requested record is made available, even if it is locked.

13) If neither an at end nor an invalid key condition occurs during the execution of a READ statement, the AT END phrase or the INVALID KEY phrase is ignored, if specified, and the following actions occur:

a) The I-O status associated with file-name-1 is updated and, if the record operation conflict condition did not occur, the file position indicator is set.

b) If an exception condition that is not an at end or an invalid key condition exists, control is transferred according to the following rules:

1. If a USE AFTER EXCEPTION procedure is associated with the file connector referenced by file-name-1, control is transferred according to the rules for the specific exception condition and the rules for the USE statement following the execution of the associated declarative procedure.

2. If there is no USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-1, control is transferred according to the rules for the specific exception condition. If the exception condition is not a fatal exception condition, control is transferred to the end of the READ statement.

c) If no exception condition exists, the record is made available in the record area and any implicit move resulting from the presence of an INTO phrase is executed. Control is transferred to the end of the READ statement, or, if the NOT AT END phrase or NOT INVALID KEY phrase is specified, to imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the READ statement.

14) If the number of bytes in the record that is read is less than the minimum size specified by the record description entries for file-name-1, the portion of the record area that is to the right of the last valid character read is undefined. If the number of bytes in the record that is read is greater than the maximum size specified by the record description entries for file-name-1, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status is set indicating a record length conflict has occurred. (See 9.1.12, I-O status.)

15) Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged; a record is available to the runtime element prior to the execution of imperative-statement-2 or imperative-statement-4, if specified, or prior to the execution of any statement following the READ statement, if imperative-statement-2 or imperative-statement-4 is not specified.

16) Unless otherwise specified, at the completion of any unsuccessful execution of a READ statement, the content of the associated record area is undefined, the key of reference is undefined for indexed files, and the file position indicator is set to indicate that no valid record position has been established.

FORMAT 1

17) An implicit or explicit NEXT phrase or a PREVIOUS phrase results in a sequential read: otherwise, the read is a random read and the rules for format 2 apply.

18) If the PREVIOUS phrase is specified, the physical file associated with the file connector referenced by file-name-1 shall be a single reel/unit mass storage file.

19) The setting of the file position indicator at the start of the execution of the READ statement is used in determining the record to be made available according to the following rules. Comparisons for records in sequential files relate to the record number. Comparisons for records in relative files relate to the relative key number. Comparisons for records in indexed files relate to the value of the current key of reference. For indexed files, the comparisons are made according to the collating sequence of the file.

a) If the file position indicator indicates that no valid record position has been established, execution of the READ statement is unsuccessful.

b) If the file position indicator indicates that an optional input file is not present, the I-O status value associated with file-name-1 is set to '10', the at end condition exists, and execution proceeds as specified in general rule 22.

c) If the file position indicator was established by a prior OPEN or START statement, the first existing record that is selected is either:

   1. If NEXT is specified or implied, the first existing record in the physical file whose record number or key value is greater than or equal to the file position indicator, or

   2. If PREVIOUS is specified, the first existing record in the physical file whose record number or key value is less than or equal to the file position indicator.

      NOTE For OPEN, this means that you normally get the first record in the file for sequential or relative and normally get an at end condition for indexed.

d) If the file position indicator was established by a prior READ statement and the file is an indexed file whose current key of reference does not allow duplicates or a sequential or relative file, the first existing record in the physical file whose record number or key value is greater than the file position indicator if NEXT is specified or implied or is less than the file position indicator if PREVIOUS is specified is selected.

e) For indexed files, if the file position indicator was established by a prior READ statement, and the current key of reference does allow duplicates, the record that is selected is one of the following:

   1. If NEXT is specified or implied,

      a. If there exists in the physical file a record whose key value is equal to the file position indicator and whose logical position within the set of duplicates is after the record that was made available by that prior READ statement, the record within the set of duplicates that is immediately after the record that was made available by that prior READ statement;

      b. otherwise; the first record in the physical file whose key value is greater than the file position indicator.

   2. If PREVIOUS is specified,

      a. If there exists in the physical file a record whose key value is equal to the file position indicator and whose logical position within the set of duplicates is before the record that was made available by that prior READ statement, the record within the set of duplicates that is immediately before the record that was made available by that prior READ statement;

      b. otherwise, the last record within the set of duplicates, if any, whose key value is the first key value less than the file position indicator.

If a record is found that satisfies this general rule and other general rules for the READ statement, the record is made available in the record area associated with file-name-1 unless the RELATIVE KEY clause is specified for file-name-1 and the number of significant digits in the relative record number of the selected record is larger than the size of the relative key data item. In that case, the I-O status value associated with file-name-1 is set to '14', the at end condition exists, the file position indicator is set to indicate that no next or previous logical record exists, and execution proceeds as specified in general rule 22.

NOTE Except in the case of an indexed file, the record made available may have a length of zero.

If no record is found that satisfies the above rules, the file position indicator is set to indicate that no next or previous logical record exists, the I-O status value associated with file-name-1 is set to '10', the at end condition exists, and execution proceeds as specified in general rule 22.

If a record is made available, the file position indicator is updated as follows:

a) For sequential files, the file position indicator is set to the record number of the record made available.

b) For relative files, the file position indicator is set to the relative record number of the record made available.

c) For indexed files, the file position indicator is set to the value of the current key of reference of the record made available.

20) If the ADVANCING ON LOCK phrase is specified on the READ statement of a file open for file sharing and the record to be made available is locked by another file connector, the result of this READ statement is as if the locked record were read and then the same READ statement were executed. If the record to be made available is locked by another file connector, this action is repeated until either an unlocked record is read or the end of the file is encountered if NEXT is specified or implied, or the beginning of file is encountered if PREVIOUS is specified. A record operation conflict condition does not exist. If the end of the file or beginning of the file is encountered, the file position indicator is set to indicate that no next or previous logical record exists and execution proceeds as indicated in general rule 22.

    If the file is not open for file sharing, the ADVANCING ON LOCK phrase is ignored.

21) If, during the execution of the READ statement, the end of reel/unit is recognized or a reel/unit contains no logical records, and the logical end of the file has not been reached for a given file connector, a reel/unit swap occurs and the current volume pointer is updated to point to the next reel/unit existing for the physical file.

22) If the at end condition exists, the following occurs in the order specified:

    a) The I-O status value associated with file-name-1 is set to '10' to indicate the at end condition.

    b) If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement-1. Any USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-1 is not executed. Execution then continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules of that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the READ statement and the NOT AT END phrase, if specified, is ignored.

    c) If the AT END phrase is not specified and a USE AFTER EXCEPTION procedure is associated with the file connector referenced by file-name-1, that procedure is executed. Control is then transferred to the end of the READ statement. The NOT AT END phrase is ignored if it is specified.

    d) If the AT END phrase is not specified and there is no USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-1, control is transferred to the end of the READ statement. The NOT AT END phrase is ignored if it is specified.

    When the at end condition exists, execution of the READ statement is unsuccessful.

23) For a relative file, if the RELATIVE KEY clause is specified for file-name-1, the execution of a READ statement moves the relative record number of the record made available to the relative key data item according to the rules for the MOVE statement.

24) For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key that is the key of reference are made available in the same order, or, in the case of PREVIOUS, in the reverse order, in which they are released by execution of WRITE statements, or by execution of REWRITE statements that create such duplicate values.

25) The I-O status for the file connector referenced by file-name-1 is set to '02' if the execution of the READ statement is successful, an indexed file is being sequentially accessed, the key of reference is an alternate record key, and one of the following is true:

a) the NEXT phrase is specified or implied and the alternate record key in the record that follows the record that was successfully read duplicates the same key in the record that was successfully read, or

b) the PREVIOUS phrase is specified and the alternate record key in the record that immediately precedes the record that was successfully read duplicates the same key in the record that was successfully read.

NOTE 1 If the sharing mode of the file is sharing with all other, I-O status value '02' on a sequential read should not be relied on for a subsequent sequential read. The record with a duplicate key might have been deleted through another file connector between the return of I-O status value '02' and the execution of the subsequent READ statement.

NOTE 2 If the sharing mode of the file is sharing with all other, the lack of an I-O status value '02' on a sequential read should not be relied on as an indication that no duplicate key will exist at the time of a subsequent sequential read. A record with a duplicate key might have been added through another file connector before the execution of that subsequent READ statement.

FORMAT 2

26) If, at the time of the execution of a READ statement, the file position indicator indicates that an optional input file is not present, the invalid key condition exists and execution of the READ statement is unsuccessful. (See 9.1.13, Invalid key condition.)

27) For a relative file, execution of a READ statement sets the file position indicator to the value contained in the data item referenced by the RELATIVE KEY clause for the file, and the record whose relative record number equals the file position indicator is made available in the record area associated with file-name-1. If the physical file does not contain such a record, the invalid key condition exists and execution of the READ statement is unsuccessful. (See 9.1.13, Invalid key condition.)

28) For an indexed file accessed through a given file connector, if the KEY phrase is specified, data-name-1 or record-key-name-1 is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of sequential format READ statements for the file through the file connector until a different key of reference is established for the file through that file connector.

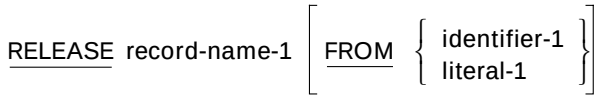
29) For an indexed file accessed through a given file connector, if the KEY phrase is not specified, the prime record key is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of sequential format READ statements for the file through the file connector until a different key of reference is established for the file through that file connector.

30) For an indexed file accessed through a given file connector, execution of a READ statement sets the file position indicator to the value in the key of reference. This value is compared with the value contained in the corresponding data item of the stored records in the file until the first record having an equal value is found. In the case of an alternate key with duplicate values, the first record found is the first record in a sequence of duplicates that was released to the operating environment. The record so found is made available in the record area associated with file-name-1. If no record is so identified, the invalid key condition exists and execution of the READ statement is unsuccessful. (See 9.1.13, Invalid key condition.)

14.9.30 RELEASE statement

The RELEASE statement transfers records to the initial phase of a sort operation.

14.9.30.1 General format



14.9.30.2 Syntax rules

1) Record-name-1 shall be the name of a logical record in a sort-merge file description entry and it may be qualified.

2) If identifier-1 is a function-identifier, it shall reference an alphanumeric or national function.

3) Identifier-1 or literal-1 shall be valid as a sending operand in a MOVE statement specifying record-name-1 as the receiving operand.

4) Literal-1 shall not be a zero-length literal.

14.9.30.3 General rules

1) A RELEASE statement may be executed only when it is within the range of an input procedure being executed by a SORT statement that references the file-name associated with record-name-1. If it is executed at any other time, the EC-FLOW-RELEASE exception condition is set to exist.

2) The execution of a RELEASE statement causes the record named by record-name-1 to be released to the initial phase of a sort operation.

3) The logical record released by the execution of the RELEASE statement is no longer available in the record area unless the sort-merge file-name associated with record-name-1 is specified in a SAME RECORD AREA clause. The logical record is also available as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as the file associated with record-name-1.

4) The result of the execution of a RELEASE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

   a) The statement:

```
MOVE identifier-1 TO record-name-1
```

   or

```
MOVE literal-1 TO record-name-1
```

   according to the rules specified for the MOVE statement.

   b) The same RELEASE statement without the FROM phrase.

NOTE 14.6.9, Overlapping operands, and 14.9.24, MOVE statement, general rules, apply to any cases in which the storage area identified by identifier-1 and the record area associated with record-name-1 share any part of their storage areas. The result of execution of the RELEASE statement is undefined if the result of execution of the implicit MOVE statement described in general rule 4b is undefined.

5) After the execution of the RELEASE statement is complete, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name-1 is not available except as specified by the SAME RECORD AREA clause.

6) If the number of bytes to be released to the sort operation is greater than the number of bytes in record-name-1, the content of the bytes that extend beyond the end of record-name-1 are undefined.

14.9.31 RESUME statement

The RESUME statement transfers control to a procedure-name or to the statement following the statement that caused a declarative to be executed.

14.9.31.1 General format

```
RESUME AT { NEXT STATEMENT  }
          { procedure-name-1 }
```

14.9.31.2 Syntax rules

1) The RESUME statement may be specified only in a declarative.

2) The RESUME statement shall not be specified in a declarative procedure for which the GLOBAL phrase is specified in the associated USE statement.

3) Procedure-name-1 shall be a procedure-name in the nondeclarative portion of the function, method, or program.

14.9.31.3 General rules

1) If the RESUME statement is executed within the scope of execution of a global declarative, it is the equivalent of the execution of a CONTINUE statement.

2) If the NEXT STATEMENT phrase is specified, control is transferred to an implicit CONTINUE statement that that is determined as follows:

    a) When an exception condition caused a declarative to be executed, the implicit CONTINUE statement immediately follows the end of the statement that was executing when control was transferred to the declarative unless general rules associated with the applicable statement specify otherwise. The applicable statement is one of the following:

        1. If the exception condition was raised within a statement within the runtime entity and was not a propagated exception condition, the applicable statement is the one in which the exception condition was raised.

        2. If the exception condition was propagated from an activated runtime entity, the applicable statement is the CALL or INVOKE statement that activated the entity, or, for an inline invocation or a function invocation, it is the statement in which the inline invocation or function invocation was specified.

        3. If the statement is contained in other statements, the applicable statement is the lowest level statement, not the containing statement.

    b) If the declarative was not executed because of an exception condition but was executed instead by a PERFORM statement in the nondeclarative portion of the source element that referenced the declarative procedure, the implicit CONTINUE statement immediately follows the last statement of the terminating procedure referenced in that PERFORM statement.

        NOTE Use of NEXT STATEMENT may cause a transfer of control to a statement that in the normal course of events would not be executed. For example IF a GO TO x ELSE GO TO y END-IF. If an exception condition was raised during the evaluation of 'a', transfer would be after the END-IF even though control normally would never be passed there.

3) If procedure-name-1 is specified, control is transferred to procedure-name-1 as if a GO TO procedure-name-1 were executed.

    NOTE Use of this method of recovery may cause the flow of control for PERFORM statements to be undefined as described in 14.9.27, PERFORM statement general rule 11.

14.9.32 RETURN statement

The RETURN statement obtains either sorted records from the final phase of a sort operation or merged records during a merge operation.

14.9.32.1 General format

```
RETURN file-name-1 RECORD [ INTO identifier-1 ]
  AT END imperative-statement-1
  [ NOT AT END imperative-statement-2 ]
  [ END-RETURN ]
```

14.9.32.2 Syntax rules

1) File-name-1 shall be described by a sort-merge file description entry in the data division.

2) The INTO phrase may be specified in a RETURN statement:

   a) If only one record description is subordinate to the sort-merge file description entry, or

   b) If all record-names associated with file-name-1 and the data item referenced by identifier-1 describe an alphanumeric group item or an elementary item of category alphanumeric or category national.

3) If identifier-1 is a strongly-typed group item, there shall be exactly one record area subordinate to the SD for file-name-1. This record area shall be a strongly-typed group item of the same type as identifier-1.

4) The AT END phrase and the NOT AT END phrase, when specified, may be written in reversed order.

14.9.32.3 General rules

1) A RETURN statement may be executed only when it is within the range of an output procedure being executed by a MERGE or SORT statement that references file-name. If it is executed at any other time, the EC-FLOW-RETURN exception condition is set to exist.

2) When the logical records of a file are described with more than one record description, the content of any data item, where any part of that data item lies beyond the range of the current record, is undefined at the completion of the execution of the RETURN statement.

3) The execution of the RETURN statement causes the next existing record in the file referenced by file-name-1, as determined by the keys listed in the SORT or MERGE statement, to be made available in the record area associated with file-name-1. If no next logical record exists in the file referenced by file-name-1, the at end condition is set to exist and control is transferred to imperative-statement-1 of the AT END phrase. Execution continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred according to the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the RETURN statement and the NOT AT END phrase is ignored, if specified. When the at end condition exists, execution of the RETURN statement is unsuccessful and the contents of the record area associated with file-name-1 are undefined. After the execution of imperative-statement-1 in the AT END phrase, no RETURN statement may be executed as part of the current output procedure. If such a RETURN statement is executed, the EC-SORT-MERGE-RETURN exception condition is set to exist and the results of the execution of the RETURN statement are undefined.

4) If an at end condition does not exist during the execution of a RETURN statement, then after the record is made available and after executing any implicit move resulting from the presence of an INTO phrase, control is transferred to imperative-statement-2, if specified; otherwise, control is transferred to the end of the RETURN statement.

5) The result of the execution of a RETURN statement with the INTO phrase is equivalent to the application of the following rules in the order specified:

   a) The same RETURN statement without the INTO phrase is executed.

   b) The current record is moved from the record area to the area specified by identifier-1 according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is determined by rules specified for the RECORD clause. If the file description entry contains a RECORD IS VARYING clause, the implied move is an alphanumeric group move. The implied MOVE statement does not occur if the execution of the RETURN statement was unsuccessful. Item identification of the data item referenced by identifier-1 is done after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data item referenced by identifier-1.

      NOTE 14.6.9, Overlapping operands, and 14.9.24, MOVE statement, general rules, apply to any cases in which the storage area identified by identifier-1 and the record area associated with file-name-1 share any part of their storage areas. The result of execution of the RETURN statement is undefined if the result of execution of the implicit MOVE statement described in general rule 4b is undefined.

14.9.33 REWRITE statement

The REWRITE statement logically replaces a record existing in a mass storage file.

14.9.33.1 General format

```
REWRITE { record-name-1      } RECORD [ FROM { identifier-1 } ]
        { FILE file-name-1   }          [      { literal-1    } ]
    [ retry-phrase ]
    [ WITH LOCK    ]
    [ WITH NO LOCK ]
    [ INVALID KEY imperative-statement-1     ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-REWRITE ]
```

where retry-phrase is described in 14.7.8, RETRY phrase

14.9.33.2 Syntax rules

1) Record-name-1 is the name of a logical record in the file section of the data division and may be qualified.

2) Neither the INVALID KEY phrase nor the NOT INVALID KEY phrase shall be specified for a REWRITE statement that references a file with sequential organization or a file with relative organization and sequential access mode.

3) If record-name-1 is defined in a containing program and is referenced in a contained program, the file description entry for the file associated with record-name-1 shall contain a GLOBAL clause.

4) If automatic locking has been specified for the rewrite file, neither the WITH LOCK phrase nor the WITH NO LOCK phrase shall be specified.

5) If record-name-1 is specified, identifier-1 or literal-1 shall be valid as a sending operand in a MOVE statement specifying record-name-1 as the receiving operand.

6) If identifier-1 references a bit data item other than an intrinsic function and the FILE phrase is specified, identifier-1 shall be described such that:

   a) subscripting and reference modification in identifier-1 consist of only fixed-point numeric literals or arithmetic expressions in which all operands are fixed-point numeric literals and the exponentiation operator is not specified; and

   b) it is aligned on a byte boundary.

7) If identifier-1 references an intrinsic function and the FILE phrase is specified, identifier-1 shall reference an alphanumeric or national function.

8) If identifier-1 references an intrinsic function and the FILE phrase is not specified, identifier-1 shall reference an alphanumeric, boolean, or national function.

9) If the FILE phrase is specified, the FROM phrase shall also be specified and:

   a) identifier-1 shall be valid as a sending operand in a MOVE statement;

b) literal-1 shall be an alphanumeric, boolean, or national literal and shall not be a figurative constant.

10) File-name-1 shall not reference a report file or a sort-merge file description entry.

11) If the FILE phrase is specified, the description of identifier-1, including its subordinate data items, shall not contain a data item described with a USAGE OBJECT REFERENCE clause.

14.9.33.3 General rules

1) The rewrite file connector is the file connector referenced by file-name-1 or the file-name associated with record-name-1.

2) The rewrite file connector shall have an open mode of I-O. If the open mode is some other value or the file is not open, the I-O status in the rewrite file connector is set to '49' and the execution of the REWRITE statement is unsuccessful.

3) The successful execution of the REWRITE statement releases a logical record to the operating environment.

4) If the rewrite file connector has an access mode of sequential, the immediately previous input-output statement executed that referenced this file connector shall have been a successfully executed READ statement. If this is not true, the I-O status in the rewrite file connector is set to '43' and the execution of the REWRITE statement is unsuccessful. For a successful REWRITE statement, the operating environment logically replaces the record that was accessed by the READ statement.

   NOTE Logical records in relative and sequential files may have a length of zero. Logical records in an indexed file shall always be long enough to contain the record keys.

5) The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless file-name-1 or the file-name associated with record-name-1 is specified in a SAME RECORD AREA clause. The logical record is also available as a record of other file-names referenced in the same SAME RECORD AREA clause as file-name-1 or the file-name associated with record-name-1, as well as the file associated with record-name-1.

6) The result of the execution of a REWRITE statement specifying record-name-1 and the FROM phrase is equivalent to the execution of the following statements in the order specified:

   a) The statement:

      ```
      MOVE identifier-1 TO record-name-1
      ```

      or

      ```
      MOVE literal-1 TO record-name-1
      ```

      according to the rules specified for the MOVE statement.

   b) The same REWRITE statement without the FROM phrase.

   NOTE 14.6.9, Overlapping operands, and 14.9.24, MOVE statement, general rules, apply to any cases in which the storage area identified by identifier-1 and the record area associated with file-name-1 share any part of their storage areas. The result of execution of the REWRITE statement is undefined if the result of execution of the implicit MOVE statement described in general rule 6a is undefined.

7) The figurative constant SPACE when specified in the REWRITE statement references one alphanumeric space character.

8) The result of execution of a REWRITE statement with the FILE phrase is equivalent to the execution of the following in the order specified:

— The statement

```
MOVE identifier-1 TO implicit-record-1
```

or

```
MOVE literal-1 TO implicit-record-1
```

— The statement

```
REWRITE implicit-record-1
```

where implicit-record-1 refers to the record area for file-name-1 and is treated:

a) when identifier-1 references an intrinsic function, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same class, category, usage, and length as the returned value of the intrinsic function, or

b) when identifier-1 does not reference an intrinsic function, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same description as identifier-1, or

c) when literal-1 is specified, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same class, category, usage, and length as literal-1.

NOTE 14.6.9, Overlapping operands, and 14.9.24, MOVE statement, general rules, apply to any cases in which the storage area identified by identifier-1 and the record area associated with implicit-record-1 share any part of their storage areas. The result of execution of the REWRITE statement is undefined if the result of execution of the implicit MOVE statement described in general rule 4b is undefined.

9) After the execution of the REWRITE statement is complete, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name-1 is not available except as specified for the SAME RECORD AREA clause as indicated in general rule 5.

10) If record locking is enabled for the rewrite file connector and the record identified for rewriting is locked by another file connector, the result of the operation depends on the presence or absence of the RETRY phrase. If the RETRY phrase is specified, additional attempts may be made to rewrite the record as specified in the rules in 14.7.8, RETRY phrase. If the RETRY phrase is not specified or the record is not successfully rewritten as specified by the RETRY phrase, the record operation conflict condition exists. The I-O status is set in accordance with the rules for the RETRY phrase. When the record operation conflict condition exists as a result of the REWRITE statement:

a) The file position indicator is unchanged.

b) A value is placed into the I-O status associated with the rewrite file connector to indicate the record operation conflict condition.

c) The REWRITE statement is unsuccessful.

11) If record locks are in effect, the following actions take place at the beginning or at the successful completion of the execution of the REWRITE statement:

a) If single record locking is specified for the rewrite file connector:

1. If that file connector holds a record lock on the record to be logically replaced, that lock is released at completion unless the WITH LOCK phrase is specified.

2. If that file connector holds a record lock on a record other than the one to be logically replaced, that lock is released at the beginning.

b) If multiple record locking is specified for the rewrite file connector, and a record lock is associated with the record to be logically replaced, that record lock is released at completion only when the WITH NO LOCK phrase is specified and the record to be logically replaced was already locked by that file connector.

c) If the WITH LOCK phrase is specified, the record lock associated with the record to be replaced is set at completion.

12) The file position indicator in the rewrite file connector is not affected by the execution of a REWRITE statement.

13) The execution of the REWRITE statement causes the I-O status value in the rewrite file connector to be updated as indicated in 9.1.12, I-O status.

14) If the execution of the REWRITE statement is unsuccessful, no logical record updating takes place, the content of the record area is unaffected, and the I-O status in the rewrite file connector is updated as indicated in other general rules.

15) When record-name-1 is specified, if the number of bytes to be written to the file is greater than the number of bytes in record-name-1, the content of the bytes that extend beyond the end of record-name-1 are undefined.

SEQUENTIAL FILES

16) If the number of bytes in the data item referenced by identifier-1, the runtime representation of literal-1, or the record referenced by record-name-1 is not equal to the number of bytes in the record being replaced, the execution of the REWRITE statement is unsuccessful and the I-O status in the rewrite file connector is set to '44'.

RELATIVE AND INDEXED FILES

17) The number of bytes in the record referenced by identifier-1, the runtime representation of literal-1, or the record referenced by record-name-1 may differ from the number of bytes in the record being replaced.

18) Transfer of control following the successful or unsuccessful execution of the REWRITE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the REWRITE statement. (See 9.1.13, Invalid key condition.)

19) The number of bytes in the runtime representation of literal-1, the data item referenced by identifier-1, or the record referenced by record-name-1 after any changes made to the record length by the FORMAT clause shall not be larger than the largest or smaller than the smallest number of bytes allowed by the RECORD IS VARYING clause associated with file-name-1 or the file-name associated with record-name-1. If this rule is violated, the execution of the REWRITE statement is unsuccessful and the I-O status in the rewrite file connector is set to '44'.

RELATIVE FILES

20) For a file accessed in either random or dynamic access mode, the operating environment logically replaces the record identified by the relative key data item specified for file-name-1 or the file-name associated with record-name-1. If the file does not contain the record specified by the key, the invalid key condition exists. When the invalid key condition is recognized, the execution of the REWRITE statement is unsuccessful and the I-O status in the rewrite file connector is set to the invalid key condition '23'.

INDEXED FILES

21) If the access mode of the REWRITE file connector is sequential, the record to be replaced is specified by the value of the prime record key. When the REWRITE statement is executed the value of the prime record key of the record to be replaced shall be equal to the value of the prime record key of the last record read using this file connector. If it is not, the execution of the REWRITE statement is unsuccessful and the I-O status in the rewrite file connector is set to the invalid key condition, '21'.

22) If the access mode of the rewrite file connector is random or dynamic, the record to be replaced is specified by the prime record key. If there is no existing record in the physical file with that prime record key, the execution of the REWRITE statement is unsuccessful and the I-O status in the rewrite file connector is set to the invalid key condition, '23'.

23) Execution of the REWRITE statement for a record that has an alternate record key occurs as follows:

    a) When the value of a specific alternate record key is not changed, the order of retrieval when that key is the key of reference remains unchanged.

    b) When the value of a specific alternate record key is changed, the subsequent order of retrieval of that record may be changed when that specific alternate record key is the key of reference. When duplicate key values are permitted, the record is logically positioned last within the set of duplicate records where the alternate record key value is equal to the same alternate key value in one or more records in the file based on the collating sequence for the file.

        NOTE If two or more file connectors share the physical file, a duplicate alternate key might not actually be positioned last at the completion of the REWRITE statement, because another duplicate key might have been created by another operation.

24) The comparison for equality for record keys is based on the collating sequence for the file according to the rules for a relation condition. The invalid key condition exists under the following circumstances:

    a) When the rewrite file connector is open in the sequential access mode and the value of the prime record key of the record to be replaced is not equal to the value of the prime record key of the last record read through the file connector, the I-O status associated with the file connector is set to '21'.

    b) When the rewrite file connector is open in the dynamic or random access mode and the value of the prime record key of the record to be replaced is not equal to the value of the prime record key of any record existing in that physical file, the I-O status associated with the rewrite file connector is set to '23'.

    c) When an alternate record key of the record to be replaced does not allow duplicates and the value of that alternate record key is equal to the value of the corresponding alternate record key of a record in that physical file, the I-O status associated with the rewrite file connector is set to '22'.

    When the invalid key condition is recognized, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, and the content of the record area is unaffected.

14.9.34 SEARCH statement

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the value of the associated index to indicate that table element.

14.9.34.1 General format

Format 1 (serial):

```
SEARCH identifier-1 [ VARYING { identifier-2  } ]
                              { index-name-1  }
    [ AT END imperative-statement-1 ]
    { WHEN condition-1 { imperative-statement-2 } } ...
                       { NEXT SENTENCE          }
    [ END-SEARCH ]
```

NOTE NEXT SENTENCE is an archaic feature and its use should be avoided.

Format 2 (all):

```
SEARCH ALL identifier-1 [ AT END imperative-statement-1 ]
    WHEN { data-name-1 { IS EQUAL TO } { identifier-3 }   }
         {             { IS =       } { literal-1    }   }
         {                            { arithmetic-expression-1 } }
         { condition-name-1                                       }
    [ AND { data-name-2 { IS EQUAL TO } { identifier-4 }   } ]...
          {             { IS =       } { literal-2    }   }
          {                            { arithmetic-expression-2 } }
          { condition-name-2                                       }
    { imperative-statement-2 }
    { NEXT SENTENCE          }
    [ END-SEARCH ]
```

NOTE NEXT SENTENCE is an archaic feature and its use should be avoided.

14.9.34.2 Syntax rules

ALL FORMATS

1) Identifier-1 shall not be subscripted or reference modified and its data description entry shall contain an OCCURS clause including an INDEXED phrase.

2) If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase shall not be specified.

FORMAT 1

3) Identifier-2 shall reference a data item whose usage is index or a data item that is an integer. Identifier-2 shall not be subscripted by the first or only index-name specified in the INDEXED phrase in the OCCURS clause specified in the data description entry for identifier-1.

4) Condition-1 may be any conditional expression evaluated as specified in 8.8.4, Conditional expressions.

FORMAT 2

5) The OCCURS clause associated with identifier-1 shall contain the KEY phrase.

6) Data-name-1 and all repetitions of data-name-2 may be qualified, shall be subscripted by the first index-name associated with identifier-1 along with any subscripts required to uniquely identify the data item, and shall be referenced in the KEY phrase in the OCCURS clause associated with identifier-1. The index-name subscript shall not be followed by a '+' or a '–'.

7) All referenced condition-names shall be defined as having only a single value and shall be subscripted by the first index-name associated with identifier-1, along with any subscripts required to uniquely identify the condition-name. The data-name associated with each condition-name shall be specified in the KEY phrase in the OCCURS clause associated with identifier-1. The index-name subscript shall not be followed by a '+' or a '–'.

8) Identifier-3, identifier-4, identifiers specified in arithmetic-expression-1, and identifiers specified in arithmetic-expression-2 shall be neither referenced in the KEY phrase of the OCCURS clause associated with identifier-1 nor subscripted by the first index-name associated with identifier-1.

9) When a data-name in the KEY phrase in the OCCURS clause associated with identifier-1 is referenced or when a condition-name associated with a data-name in the KEY phrase in the OCCURS clause associated with identifier-1 is referenced, all preceding data-names in that KEY phrase or their associated condition-names shall also be referenced.

10) Data-name-1, data-name-2, identifier-3, or identifier-4 shall not specify a variable-length group.

11) Neither iteral-1 nor literal-2 shall be zero-length literals.

14.9.34.3 General rules

ALL FORMATS

1) The SEARCH statement automatically varies the first or only index associated with identifier-1 and tests conditions specified in WHEN phrases in the SEARCH statement to determine whether a table element satisfies these conditions. Any subscripting specified in a WHEN phrase is evaluated each time the conditions in that WHEN phrase are evaluated. For Format 1, an additional index or data item may be varied. If identifier-1 references a data item that is subordinate to a data item whose data description entry contains an OCCURS clause, only the setting of an index associated with identifier-1 (and any data item referenced by identifier-2 or any index referenced by index-name-1, if specified) is modified by the execution of the SEARCH statement. The subscript that is used to determine the occurrence of each superordinate table to search is specified by the user in the WHEN phrases. Therefore, each appropriate subscript shall be set to the desired value before the SEARCH statement is executed.

   Upon completion of the search operation, one of the following occurs:

   a) If the search operation is successful according to the general rules that follow, then: the search operation is terminated immediately; the index being varied by the search operation remains set at the occurrence number that caused a WHEN condition to be satisfied; if the WHEN phrase contains the NEXT SENTENCE phrase, control is transferred to an implicit CONTINUE statement immediately preceding the next separator period; if the WHEN phrase contains imperative-statement-2, control is transferred to that imperative-statement-2 and execution continues according to the rules for each statement specified in that

imperative-statement-2. If the execution of a procedure branching or conditional statement results in an explicit transfer of control, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of that imperative-statement-2, control is transferred to the end of the SEARCH statement.

b) If the search operation is unsuccessful according to the general rules that follow, then:

1. If the AT END phrase is specified, control is transferred to imperative-statement-1 and execution continues according to the rules for each statement specified in imperative-statement-1. If the execution of a procedure branching or conditional statement results in an explicit transfer of control, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the SEARCH statement.

2. If the AT END phrase is not specified and either the EC-RANGE-SEARCH-INDEX or EC-RANGE-SEARCH-NO-MATCH exception condition was raised during the execution of the SEARCH statement and an applicable declarative exists, control is transferred to that declarative and, if control is returned from that declarative control is transferred to the end of the SEARCH statement.

3. If the AT END phrase is not specified and neither exception condition was raised because the checking for those exception conditions was not enabled, control is transferred to the end of the SEARCH statement.

2) The comparison associated with each WHEN phrase is executed in accordance with the rules specified for conditional expressions. (See 8.8.4, Conditional expressions.)

FORMAT 1

3) The index to be varied by the search operation is referred to as the search index and it is determined as follows:

a) If the VARYING phrase is not specified, the search index is the index referenced by the first (or only) index-name specified in the INDEXED phrase in the OCCURS clause associated with identifier-1.

b) If the VARYING identifier-2 phrase is specified, the search index is the same as in general rule 3a and the following also applies:

1. If identifier-2 references an index data item, that data item is incremented by the same amount as, and at the same time as, the search index.

2. If identifier-2 references an integer data item, that data item is incremented by the value one at the same time as the search index is incremented.

c) If the VARYING index-name-1 phrase is specified, the search index depends on the following:

1. If index-name-1 is specified in the INDEXED BY phrase in the OCCURS clause associated with identifier-1, the index referenced by index-name-1 is the search index.

2. If index-name-1 is not one of the indexes specified in the INDEXED phrase in the OCCURS clause associated with identifier-1, the search index is the same as in general rule 3a. The index referenced by index-name-1 is incremented by one occurrence number at the same time as the search index is incremented.

Only the data item and indexes indicated are varied by the search operation. All other indexes associated with identifier-1 are unchanged by the search operation.

4) The search operation is serial, starting from the occurrence number that corresponds to the value of the search index at the beginning of the execution of the SEARCH statement. If, at the start of the execution, the search index contains a value that corresponds to an occurrence number that is negative, zero, or greater than the highest permissible occurrence number for identifier-1, the search operation is unsuccessful, the

EC-RANGE-SEARCH-INDEX exception condition is set to exist, and execution proceeds as indicated in general rule 1b. The number of occurrences of identifier-1, the last of which is permissible, is specified in the OCCURS clause. If, at the start of the execution of the SEARCH statement, the search index contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1, the search operation proceeds by evaluating the conditions in the order they are written. If none of the conditions is satisfied, the search index is incremented by one occurrence number. The process is then repeated using the new index setting unless the new value for the search index corresponds to a table element outside the permissible range of occurrence values, in which case the search operation is unsuccessful, the EC-RANGE-SEARCH-NO-MATCH exception condition is set to exist, and execution proceeds as indicated in general rule 1b. If one of the conditions is satisfied upon its evaluation, the search operation is successful and the execution proceeds as indicated in general rule 1a.

FORMAT 2

5) At the start of the execution of a SEARCH statement with the ALL phrase specified, the following conditions shall be true:

   a) The contents of each key data item referenced in the WHEN phrase shall be sequenced in the table according to the ASCENDING or DESCENDING phrase associated with that key data item. (See 13.18.37, OCCURS clause.)

   b) If identifier-1 is subordinate to one or more data description entries that contain an OCCURS clause, the evaluation of the conditions within a WHEN phrase that reference a key data item subordinate to identifier-1 shall result in the same occurrence number for any subscripts associated with a given level of the superordinate tables. That is, the outermost level occurrence numbers shall all be equal, the next level occurrence numbers shall all be equal down to, but not including, the innermost table.

6) If any condition specified in general rule 5 is not satisfied:

   a) If one or more settings of the search index satisfy all conditions in the WHEN phrase, one of the following occurs:

      1. the final setting of the search index is set equal to one of those settings, but it is undefined which one; execution proceeds as in general rule 1a;

      2. the final setting of the search index is undefined, the EC-RANGE-SEARCH-NO-MATCH exception condition is set to exist, and execution proceeds as in general rule 1b.

      It is undefined which of these alternatives occurs.

   b) If no such setting of the search index exists, the final setting of the search index is undefined, the EC-RANGE-SEARCH-NO-MATCH exception condition is set to exist, and execution proceeds as in general rule 1b.

7) If both conditions specified in general rule 5 are satisfied and there is more than one setting of the search index for which all conditions in the WHEN phrase are satisfied, the search operation is successful. The final setting of the search index is equal to one of them, but it is undefined which one.

8) The search index is the index referenced by the first (or only) index-name specified in the INDEXED phrase in the OCCURS clause associated with identifier-1. Any other indexes associated with identifier-1 remain unchanged by the search operation.

9) A nonserial type of search operation may take place. The initial setting of the search index is ignored. Its setting is varied during the search operation in a manner specified by the implementor. At no time is it set to a value that exceeds the value that corresponds to the last element of the table or is less than the value that corresponds to the first element of the table. The length of the table is discussed in the OCCURS clause. If any of the conditions specified in the WHEN phrase is not satisfied for any setting of the search index within the permitted range, the final setting of the search index is undefined, the search operation is unsuccessful, the

EC-RANGE-SEARCH-NO-MATCH exception condition is set to exist, and execution proceeds as indicated in general rule 1b. If all the conditions are satisfied, the search operation is successful and execution proceeds as indicated in general rule 1a.

14.9.35 SET statement

The SET statement provides a means for:

- establishing reference points for table handling operations by setting indexes associated with table elements,
- altering the status of external switches,
- altering the value of conditional variables,
- assigning object references,
- altering the attributes associated with a screen item,
- assigning the address of a data item to a data-pointer data item,
- assigning the address of a function to a function-pointer data item,
- assigning the address of a based item,
- assigning the address of a program to a program-pointer data item,
- setting and saving locale categories, and
- clearing the last exception status.

14.9.35.1 General format

Format 1 (index-assignment):

```
SET { index-name-1 } ... TO { arithmetic-expression-1 }
    { identifier-1 }       { index-name-2           }
                           { identifier-2           }
```

Format 2 (index-arithmetic):

```
SET { index-name-3 } ... { UP BY   } arithmetic-expression-2
                         { DOWN BY }
```

Format 3 (switch-setting):

```
SET { { mnemonic-name-1 } ... TO { ON  } } ...
    {                            { OFF } }
```

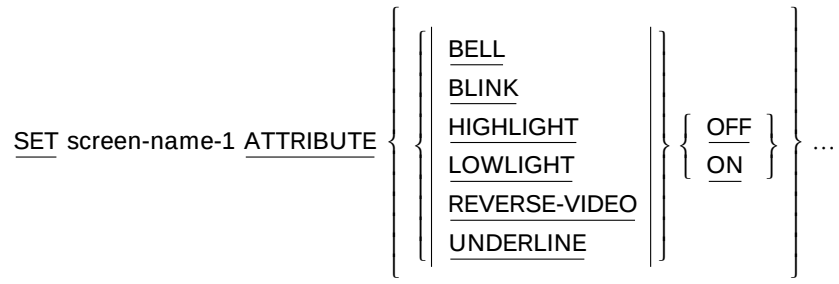
Format 4 (condition-setting):

```
SET { { condition-name-1 } ... TO { TRUE  } } ...
    {                             { FALSE } }
```

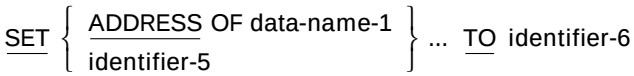
Format 5 (object-reference-assignment):

```
SET { identifier-3 } ... TO { class-name-1 }
                            { identifier-4 }
```

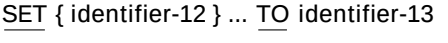
Format 6 (attribute):



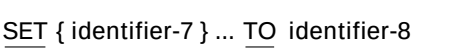
Format 7 (data-pointer-assignment):



Format 8 (function-pointer-assignment):

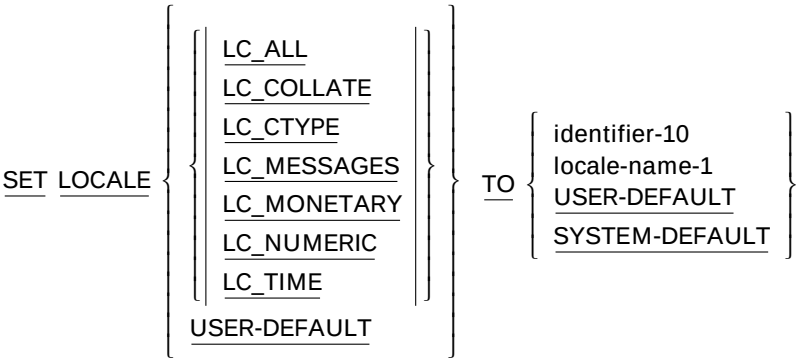


Format 9 (program-pointer-assignment):



Format 10 (data-pointer-arithmetic):

Format 11 (set-locale):



Format 12 (save-locale):

```
SET  identifier-11 TO LOCALE { LC_ALL          }
                             { USER-DEFAULT    }
```

Format 13 (saved-exception):

```
SET LAST EXCEPTION TO OFF
```

Format 14 (dynamic-capacity-table):

```
SET  date-name-2 { UP BY   } { integer-1                }
                 { DOWN BY } { arithmetic-expression-4  }
                 { TO      }
```

Format15 (IEEE-754-2008-floating-point):

```
SET CONTENT OF { identifier-14 } ... TO { NEGATIVE-INFINITY }
                                       { NOT-A-NUMBER     }
                                       { POSITIVE-INFINITY }
```

14.9.35.2 Syntax rules

FORMAT 1

1) Identifier-1 shall reference a data item of class index or an integer data item.

2) Identifier-2 shall reference a data item of class index.

3) If identifier-1 references a data item of class index, arithmetic-expression-1 shall not be specified.

4) If identifier-1 references a numeric data item, index-name-2 shall be specified.

FORMAT 3

5) Mnemonic-name-1 shall be associated with an external switch, the status of which may be altered. The implementor defines which external switches may be referenced by the SET statement.

FORMAT 4

6) Condition-name-1 shall be associated with a conditional variable.

7) If the FALSE phrase is specified, the FALSE phrase shall be specified in the VALUE clause of the data description entry for condition-name-1.

FORMAT 5

8) Identifier-3 shall be any item of class object that is permitted as a receiving item.

9) Identifier-4 shall be an object reference; the predefined object reference SUPER shall not be specified.

10) If identifier-4 is specified and the data item referenced by identifier-3 is described with an interface-name that identifies the interface int-1, the data item referenced by identifier-4 shall be one of the following:

    a) an object reference described with an interface-name that identifies int-1 or an interface inheriting from int-1;

    b) an object reference described with a class-name, subject to the following rules:

        1. if described with a FACTORY phrase, the factory object of the specified class shall implement int-1,

        2. if described without a FACTORY phrase, the objects of the specified class shall implement int-1;

    c) an object reference described with an ACTIVE-CLASS phrase, subject to the following rules:

        1. if described with a FACTORY phrase, the factory object of the class containing the data item referenced by identifier-4 shall implement int-1,

        2. if described without a FACTORY phrase, the objects of the class containing the data item referenced by identifier-4 shall implement int-1;

    d) the predefined object reference SELF, subject to the following rules:

        1. if the SET statement is contained in a method within the factory definition of the class, that factory definition shall be described with an IMPLEMENTS clause that references int-1,

        2. if the SET statement is contained in a method within the instance definition of the class, that instance definition shall be described with an IMPLEMENTS clause that references int-1;

    e) the predefined object reference NULL.

11) If class-name-1 is specified and the data item referenced by identifier-3 is described with an interface-name that identifies the interface int-1, the factory object of class-name-1 shall be described with an IMPLEMENTS clause that references int-1.

12) If identifier-4 is specified and the data item referenced by identifier-3 is described with a class-name, the data item referenced by identifier-4 shall be one of the following:

    a) an object reference described with a class-name, subject to the following rules:

        1. if the data item referenced by identifier-3 is described with an ONLY phrase, the data item referenced by identifier-4 shall be described with the ONLY phrase, and the class-name specified in the description of the data item referenced by identifier-4 shall be the same as the class-name specified in the description of the data item referenced by identifier-3,

        2. if the data item referenced by identifier-3 is described without an ONLY phrase, the class-name specified in the description of the data item referenced by identifier-4 shall reference the same class or a subclass of the class specified in the description of the data item referenced by identifier-3,

        3. the presence or absence of the FACTORY phrase shall be the same as in the description of the data item referenced by identifier-3;

    b) an object reference described with an ACTIVE-CLASS phrase, subject to the following rules:

        1. the data item referenced by identifier-3 shall not be described with the ONLY phrase,

        2. the class containing the data item referenced by identifier-4 shall be the same class or a subclass of the class specified in the description of the data item referenced by identifier-3,

3. the presence or absence of the FACTORY phrase shall be the same as in the description of the data item referenced by identifier-3;

c) the predefined object reference SELF, subject to the following rules:

1. the data item referenced by identifier-3 shall not be described with the ONLY phrase,

2. the class containing the SET statement shall be the same class or a subclass of the class specified in the description of the data item referenced by identifier-3,

3. if the data item referenced by identifier-3 is described without a FACTORY phrase, the method containing the SET statement shall be defined in the instance definition of its containing class;,

4. if the data item referenced by identifier-3 is described with a FACTORY phrase, the method containing the SET statement shall be defined in the factory definition of its containing class;

d) the predefined object reference NULL.

13) If class-name-1 is specified and the data item referenced by identifier-3 is described with a class-name, the data item shall be described with the FACTORY phrase, and the following rules apply:

a) if the data item referenced by identifier-3 is described with the ONLY phrase, class-name-1 shall be the class-name specified in the description of the data item referenced by identifier-3;

b) otherwise, class-name-1 shall reference the same class or a subclass of the class specified in the description of the data item referenced by identifier-3.

14) If the data item referenced by identifier-3 is described with an ACTIVE-CLASS phrase, the data item referenced by identifier-4 shall be one of the following:

a) an object reference described with the ACTIVE-CLASS phrase, where the presence or absence of the FACTORY phrase is the same as in the data item referenced by identifier-3;

b) the predefined object SELF, subject to the following rules:

1. if the data item referenced by identifier-3 is described without a FACTORY phrase, the method containing the SET statement shall be defined in the instance definition of its containing class,

2. if the data item referenced by identifier-3 is described with a FACTORY phrase, the method containing the SET statement shall be defined in the factory definition of its containing class;

c) the predefined object reference NULL.

FORMAT 6

15) Any particular screen-attribute shall not be specified more than once in a SET statement.

16) HIGHLIGHT and LOWLIGHT shall not both be specified in the same SET statement.

FORMAT 7

17) Identifier-5 shall reference a data item of category data-pointer. Identifier-6 shall be of category data-pointer.

18) Data-name-1 shall be a based data item.

19) If identifier-5 references a restricted data-pointer, identifier-6 shall be the predefined address NULL or shall reference a data-pointer restricted to the same type. If data-name-1 is a strongly-typed group item or a restricted pointer, identifier-6 shall reference a data-pointer restricted to the type of data-name-1.

If identifier-6 references a restricted data-pointer, either identifier-5 shall reference a data-pointer restricted to the same type or data-name-1 shall be a typed item of the type to which identifier-6 is restricted.

FORMAT 8

20) Identifier-12 shall reference a data item of category function-pointer. Identifier-13 shall be of category function-pointer. Identifier-13 shall be the predefined address NULL or shall reference a function-pointer. The function-prototypes associated with identifier-12 and identifier-13 shall have the same signature.

FORMAT 9

21) Identifier-7 shall reference a data item of category program-pointer. Identifier-8 shall be of category program-pointer.

22) If identifier-7 references a restricted program-pointer, identifier-8 shall be the predefined address NULL or shall reference a program-pointer and the program-prototypes associated with identifier-7 and identifier-8 shall have the same signature.

FORMAT 10

23) Identifier-9 shall be of category data-pointer.

24) Identifier-9 shall not be a data-pointer restricted to a type described with the STRONG phrase.

FORMAT 11

25) If USER-DEFAULT is specified as the first operand, identifier-10 or locale-name-1 shall be specified in the TO phrase.

26) Locale-name-1 shall be specified in the LOCALE clause of the SPECIAL-NAMES paragraph.

27) Identifier-10 shall reference an elementary data item of category data-pointer.

FORMAT 12

28) Identifier-11 shall reference an elementary data item of category data-pointer.

FORMAT 14

29) Data-name-2 shall reference a data item defined in the CAPACITY phrase of a dynamic-capacity-table format OCCURS clause.

30) Integer-1 shall be nonnegative and, if TO is specified, integer-1 shall be not less than the minimum capacity defined in the corresponding OCCURS clause and not greater than the expected capacity, if specified.

FORMAT 15

31) Identifier-14 shall reference a data item described with an IEEE floating-point usage.

14.9.35.3 General rules

FORMATS 1 AND 2

1) Index-names are associated with a given table by being specified in the INDEXED BY phrase of the OCCURS clause for that table.

FORMAT 1

2) The value of the sending operand is determined once at the beginning of the execution of the statement. However, item identification of the data item referenced by identifier-1 is done immediately before the value of that data item is changed. For each occurrence of index-name-1 or identifier-1:

    a) If index-name-1 is specified:

        1. If arithmetic-expression-1 is specified,

            a. If the value of arithmetic-expression-1 does not result in an integer,

                —the EC-BOUND-SUBSCRIPT exception condition is set to exist,

                —the execution of the SET statement is unsuccessful and

                —the content of the receiving operand is unchanged.

            b. Otherwise, if the value of the arithmetic-expression-1 is outside the limit specified in general rule 2 of 13.18.37, OCCURS clause:

                —the EC-RANGE-INDEX exception condition is set to exist,

                —the execution of the SET statement is unsuccessful and

                —the content of the receiving operand is unchanged.

            c. Otherwise, the evaluation of the expression is successful and index-name-1 is set to a value causing it to refer to the table element that would correspond in occurrence number to that expression, even if that occurrence is not a valid occurrence within this table.

        2. If identifier-2 is specified:

            a. If the value of the identifier-2 is outside the limit specified in general rule 2 of 13.18.37, OCCURS clause:

                —the EC-RANGE-INDEX exception condition is set to exist,

                —the execution of the SET statement is unsuccessful and

                —the content of the receiving operand is unchanged.

            b. Otherwise, index-name-1 is set to a value causing it to refer to the table element that would correspond in occurrence number to the value of identifier-2, even if that occurrence is not a valid occurrence within this table.

        3. If index-name-2 is specified:

            a. If index-name-2 is associated with the same table as index-name-1, the content of the index referenced by index-name-2 is placed in the index referenced by index-name-1 unchanged.

            b. Otherwise, if the value that corresponds to the occurrence number of the table element associated with index-name-2 is outside the limit specified in general rule 2 of 13.18.37, OCCURS clause:

                —the EC-RANGE-INDEX exception condition is set to exist,

                —the execution of the SET statement is unsuccessful and

                —the content of the receiving operand is unchanged.

c. Otherwise, index-name-1 is set to a value causing it to refer to the table element that would correspond in occurrence number to the occurrence number of the table associated with index-name-2, even if that occurrence is not a valid occurrence within this table.

b) If identifier-1 references an index data item, the content of index-name-2 or of the data item referenced by identifier-2 is placed in the index referenced by index-name-1 unchanged.

c) If identifier-1 references a numeric data item, that item is set to the occurrence number of the table element referenced by index-name-2.

FORMAT 2

3) The value of arithmetic-expression-2 is determined once at the beginning of the execution of the statement. If arithmetic-expression-2 does not evaluate to an integer:

— the EC-BOUND-SUBSCRIPT exception condition is set to exist,

— the execution of the SET statement is unsuccessful, and

— the content of the receiving operand is unchanged.

4) For each occurrence of index-name-3 one of the following occurs:

a) f the value that corresponds to the occurrence number represented by the content of index-name-3 incremented (UP BY) or decremented (DOWN BY) by the result of the evaluation of arithmetic-expression-2 is outside the limit specified in general rule 2 of 13.18.37, OCCURS clause:

— the EC-RANGE-INDEX exception condition is set to exist,

— the execution of the SET statement is unsuccessful and

— the content of the receiving operand is unchanged.

b) Otherwise, index-name 3 is set to a value causing it to refer to the table element that corresponds to the occurrence number incremented or decremented by the result of the evaluation of arithmetic-expression-2, even if that occurrence is not a valid occurrence within this table.

FORMAT 3

5) The status of each external switch associated with the specified mnemonic-name-1 is modified such that the truth value resultant from evaluation of a condition-name associated with that switch will reflect an on status if the ON phrase is specified, or an off status if the OFF phrase is specified. (See 8.8.4.1.5, Switch-status condition.)

FORMAT 4

6) If the TRUE phrase is specified, the literal in the VALUE clause associated with condition-name-1 is placed in the conditional variable according to the rules for the VALUE clause, except that when the conditional variable is an alphanumeric group item, bit group item, or national group item to which a table is subordinate, its length is determined as specified in 13.18.37, OCCURS clause. If the length of the conditional variable is zero, the SET statement leaves it unchanged. If more than one literal is specified in the VALUE clause, the conditional variable is set to the value of the first literal that appears in the VALUE clause.

7) If the FALSE phrase is specified, the literal in the FALSE phrase of the VALUE clause associated with condition-name-1 is placed in the conditional variable according to the rules for the VALUE clause, except that when the conditional variable is an alphanumeric group item, bit group item, or national group item to which a table is subordinate, its length is determined as specified in 13.18.37, OCCURS clause. If the length of the conditional variable is zero, the SET statement leaves it unchanged.

8) If multiple condition-names are specified, the results are the same as if a separate SET statement had been written for each condition-name-1 in the same order as specified in the SET statement.

FORMAT 5

9) If identifier-4 is specified, a reference to the object identified by identifier-4 is placed into each data item referenced by identifier-3 in the order specified.

10) If class-name-1 is specified, a reference to the factory object of the class identified by class-name-1 is placed into each data item referenced by identifier-3 in the order specified.

FORMAT 6

11) When the SET statement is executed, the settings of the specified attributes of the screen item referenced by screen-name-1 are turned on or off as specified. The latest settings of the attributes are used when screen-name-1 is referenced in an ACCEPT screen or DISPLAY screen statement.

FORMAT 7

12) If identifier-5 is specified, the address identified by identifier-6 is stored in each data item referenced by identifier-5 in the order specified. Item identification of the data item referenced by identifier-5 is done immediately before the value of that data item is changed

13) If data-name-1 is specified, the address identified by identifier-6 is assigned to each based item referenced by data-name-1 in the order specified.

FORMAT 8

14) The address identified by identifier-13 is stored in each data item referenced by identifier-12 in the order specified. Item identification of the data item referenced by identifier-12 is done immediately before the value of that data item is changed. If the address identified by identifier-13 is neither the predefined address NULL nor the address of a function defined with the same signature as the function referenced in the definition of identifier-13, the EC-FUNCTION-PTR-INVALID exception condition is set to exist, no data items are changed, and the execution of the SET statement is terminated.

15) The effect of the SET statement on the function whose address is being stored in the function-pointer is implementor-defined.

FORMAT 9

16) The address identified by identifier-8 is stored in each data item referenced by identifier-7 in the order specified. Item identification of the data item referenced by identifier-7 is done immediately before the value of that data item is changed.

17) The effect of the SET statement on the program whose address is being stored in the program-pointer is implementor-defined.

FORMAT 10

18) If identifier-9 contains the predefined address NULL, the EC-DATA-PTR-NULL exception condition is set to exist.

19) If arithmetic-expression-3 does not evaluate to an integer, the EC-SIZE-ADDRESS exception condition is set to exist, the execution of the SET statement is unsuccessful, and the content of identifier-9 is unchanged.

20) The address contained in each identifier-9, in the order specified, is incremented, if UP is specified, or decremented, if DOWN is specified, by the number of bytes specified by arithmetic-expression-3. Item identification of the data item referenced by identifier-9 is done immediately before the value of that data item

is changed. If this new address is outside the range of values allowed by the implementor for a data-pointer data item, the EC-RANGE-PTR exception condition is set to exist and the value of the data item referenced by identifier-9 is unchanged.

FORMAT 11

21) The content of the pointer data item referenced by identifier-10 shall reference saved locale information; otherwise, the EC-LOCALE-INVALID-PTR exception condition is set to exist and the SET statement is unsuccessful.

A reference to 'saved locale' or 'saved locale information' in the rules of COBOL is a reference to a locale and its locale category information as established with a set-locale format of the SET statement.

22) If USER-DEFAULT immediately follows the keyword LOCALE, the user default locale is set to the locale identified by the second operand.

23) If a locale category immediately follows the keyword LOCALE:

a) If locale-name-1 or identifier-10 is specified in the TO phrase, the current runtime locale for the specified category is set to that category in the saved locale identified by locale-name-1 or by the content of the data item referenced by identifier-10, respectively.

b) If USER-DEFAULT is specified in the TO phrase, the current runtime locale for the specified category is set to that category in the user default locale.

c) If SYSTEM-DEFAULT is specified, the current runtime locale for the specified category is set to that category in the system default locale.

24) If the locale specified by locale-name-1 is not available, the EC-LOCALE-MISSING exception condition is set to exist.

25) Each locale category specified remains in effect for the duration of the run unit or until another SET statement specifying that category is processed.

FORMAT 12

26) If LC_ALL is specified, the current locale is saved and a reference to the saved locale is placed into the pointer data item referenced by identifier-11.

27) If USER-DEFAULT is specified, the user default locale is saved and a reference to the saved locale is placed into the pointer data item referenced by identifier-11.

FORMAT 13

28) The predefined object reference EXCEPTION-OBJECT is set to null, and the last exception status is set to indicate no exception.

FORMAT 14

29) Arithmetic-expression-4 shall evaluate to a nonnegative integer.

30) A new capacity of the dynamic-capacity table is computed, according to the rules specified in 8.5.1.6.3.3, Explicit changes in capacity, and as follows:

a) If TO is specified, the new capacity is specified by integer-1 or arithmetic-expression-4.

b) If UP is specified, the new capacity is obtained by adding integer-1 or the value of arithmetic-expression-4 to the current capacity of the table.

c) If DOWN is specified, the new capacity is obtained by subtracting integer-1 or the value of arithmetic-expression-4 from the current capacity of the table.

If the new capacity of the table exceeds the implementor's maximum capacity for this dynamic-capacity table, the EC-BOUND-TABLE-LIMIT exception condition is set to exist and the capacity of the table is unchanged; otherwise, if an expected maximum capacity is specified for the table and the new capacity of the table exceeds that expected maximum capacity, the EC-BOUND-SET exception condition is set to exist.

If the new capacity of the table is less than the minimum capacity defined in the corresponding OCCURS clause, the new capacity of the table shall be the minimum capacity. In all other cases, the new capacity of the table shall become the current capacity.

If the new capacity of the table is greater than the previous current capacity, new occurrences are created and are initialized as described in 8.5.1.6.3.4, Implicit initialization.

31) This statement shall not be executed during the execution of a SEARCH statement referring to the same table. If this rule is violated, the EC-FLOW-SEARCH exception condition is set to exist and the SET statement is not executed.

FORMAT 15

32) The data item referenced by identifier-14 is set as follows:

a) If NOT-A-NUMBER is specified, the content is set to the canonical positive signaling representation of NaN, with a payload of zero, as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings, for IEEE binary floating-point encodings, and 3.5, Decimal interchange format encodings, as limited to decimal significand encodings, for IEEE decimal floating-point encodings

b) If NEGATIVE-INFINITY is specified, the content is set to the canonical representation of negative infinity as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings, for IEEE binary floating-point encodings, and 3.5, Decimal interchange format encodings, as limited to decimal significand encodings.

c) If POSITIVE-INFINITY is specified, the content is set to the canonical representation of positive infinity as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings, for IEEE binary floating-point encodings, and 3.5, Decimal interchange format encodings, as limited to decimal significand encodings.

14.9.36 SORT statement

The SORT statement causes a set of records or table elements to be arranged in a user-specified sequence.

14.9.36.1 General format

Format 1 (file):

```
SORT file-name-1 { ON { ASCENDING } KEY { data-name-1 } ... } ...
                  { DESCENDING }
    [ WITH DUPLICATES IN ORDER ]
    [ COLLATING SEQUENCE { IS alphabet-name-1 [ alphabet-name-2 ]                    } ]
                         { { FOR ALPHANUMERIC IS alphabet-name-1 [ ] }               }
                         { { FOR NATIONAL IS alphabet-name-2     [ ] }               }
    { INPUT PROCEDURE IS procedure-name-1 [ { THROUGH } procedure-name-2 ] }
                                          { THRU    }                     ]
    { USING { file-name-2 } ...                                           }
    { OUTPUT PROCEDURE IS procedure-name-3 [ { THROUGH } procedure-name-4 ] }
                                           { THRU    }                     ]
    { GIVING { file-name-3 } ...                                          }
```

Format 2 (table):

```
SORT data-name-2 [ ON { ASCENDING } KEY [ data-name-1 ] ... ] ...
                      { DESCENDING }
    [ WITH DUPLICATES IN ORDER ]
    [ COLLATING SEQUENCE { IS alphabet-name-1 [ alphabet-name-2 ]                    } ]
                         { { FOR ALPHANUMERIC IS alphabet-name-1 [ ] }               }
                         { { FOR NATIONAL IS alphabet-name-2     [ ] }               }
```

14.9.36.2 Syntax rules

ALL FORMATS

1) A SORT statement may appear anywhere in the procedure division except that a format 1 SORT statement shall not appear in the declarative portion.

2) Alphabet-name-1 shall reference an alphabet that defines an alphanumeric collating sequence.

3) Alphabet-name-2 shall reference an alphabet that defines a national collating sequence.

FORMAT 1

4) File-name-1 shall be described in a sort-merge file description entry in the data division.

5) If the USING phrase is specified and the file description entry for file-name-1 describes variable-length records, the file description entry for file-name-2 shall describe neither records smaller than the smallest record nor larger than the largest record described for file-name-1. If the file description entry for file-name-1 describes fixed-length records, the file description entry for file-name-2 shall not describe a record that is larger than the record described for file-name-1.

6) Data-name-1 is a key data-name. Key data-names are subject to the following rules:

   a) The data items identified by key data-names shall be described in records associated with file-name-1.

   b) Key data items may be qualified.

   c) Key data items shall not be of the class boolean, object, or pointer.

   d) A key data item shall not be a variable-length group, an occurs-depending-on data item, an any-length elementary item or an item subordinate to a dynamic-capacity table.

   e) If file-name-1 has more than one record description, then the data items identified by key data-names need be described in only one of the record descriptions. The same byte positions that are referenced by a key data-name in one record description entry are taken as the key in all records of the file.

   f) None of the data items identified by key data-names may be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

   g) If the file referenced by file-name-1 contains variable-length records, all the data items identified by key data-names shall be contained within the first x bytes of the record, where x is the number of bytes of the minimum record size for the file referenced by file-name-1.

7) The words THROUGH and THRU are equivalent.

8) File-name-2 and file-name-3 shall be described in a file description entry that is not for a report file and is not a sort-merge file description entry.

9) If file-name-3 references an indexed file, the first specification of data-name-1 shall be associated with an ASCENDING phrase and the data item referenced by that data-name-1 shall begin at the same byte location within its record and occupy the same number of bytes as the prime record key for that file.

10) No pair of file-names in the same SORT statement may be specified in the same SAME SORT AREA or SAME SORT-MERGE AREA clause. File-names associated with the GIVING phrase shall not be specified in the same SAME AREA clause.

11) If the GIVING phrase is specified and the file description entry for file-name-3 describes variable-length records, the file description entry for file-name-1 shall describe neither records smaller than the smallest record nor larger than the largest record described for file-name-3. If the file description entry for file-name-3 describes fixed-length records, the file description entry for file-name-1 shall not describe a record that is larger than the record described for file-name-3.

12) If file-name-2 references a relative or an indexed file, its access mode shall be sequential or dynamic.

FORMAT 2

13) Data-name-2 may be qualified and shall have an OCCURS clause in its data description entry. Subscripting shall be specified in accordance with 8.4.1.2, Subscripts.

14) Data-name-1 is a key data-name, subject to the following rules:

    a) The data item identified by a key data-name shall be the same as, or subordinate to, the data item referenced by data-name-2.

    b) Key data items may be qualified.

    c) Key data items shall not be of class boolean, object, or pointer.

    d) A key data item shall not reference a variable-length group or an occurs-depending group item.

    e) If the data item identified by a key data-name is subordinate to data-name-2, it shall not be described with an OCCURS clause, and it shall not be subordinate to an entry that is also subordinate to data-name-2 and contains an OCCURS clause.

15) The KEY phrase may be omitted only if the description of the table referenced by data-name-2 contains a KEY phrase.

14.9.36.3 General rules

ALL FORMATS

1) The words ASCENDING and DESCENDING are transitive across all occurrences of data-name-1 until another word ASCENDING or DESCENDING is encountered.

2) The data items referenced by the specifications of data-name-1 are the key data items that determine the order in which records are returned from the file referenced by file-name-1 or the order in which the table elements are stored after sorting takes place. The order of significance of the keys is the order in which they are specified in the SORT statement, without regard to their association with ASCENDING or DESCENDING phrases.

3) If the DUPLICATES phrase is specified and the contents of all the key data items associated with one record or table element are equal to the contents of the corresponding key data items associated with one or more other records or table elements, the order of return of these records or the relative order of the contents of these table elements is:

    a) The order of the associated input files as specified in the SORT statement. Within a given input file the order is that in which the records are accessed from that file.

    b) The order in which these records are released by an input procedure, when an input procedure is specified.

    c) The relative order of the contents of these table elements before sorting takes place.

4) If the DUPLICATES phrase is not specified and the contents of all the key data items associated with one record or table element are equal to the contents of the corresponding key data items associated with one or more other records or table elements, the order of return of these records or the relative order of the contents of these table elements is undefined.

5) The alphanumeric collating sequence that applies to the comparison of key data items of class alphabetic and class alphanumeric, and the national collating sequence that applies to the comparison of key data items of class national, are each separately determined at the beginning of the execution of the SORT statement in the following order of precedence:

    a) First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in this SORT statement. The collating sequence associated with alphabet-name-1 applies to key data items of class

alphabetic and alphanumeric; the collating sequence associated with alphabet-name-2 applies to key data items of class national.

b) Second, the collating sequences established as the program collating sequences.

6) Additional rules affecting the execution of the SORT statement are given in 12.4.4, File control entry, general rules 3 and 4.

FORMAT 1

7) If the file referenced by file-name-1 contains only fixed-length records, any record in the file referenced by file-name-2 containing fewer character positions than that fixed-length is space filled on the right to that fixed length, beginning with the first character position after the last character in the record, when that record is released to the file referenced by file-name-1, as follows:

a) If there is only one record description entry associated with the file referenced by file-name-2 and that record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.

b) If there are multiple record description entries associated with the file referenced by file-name-2 and the descriptions include a SELECT WHEN clause, the rules of the SELECT WHEN clause are applied to the record to select its description. When the record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.

c) Otherwise, the record is space filled with alphanumeric space characters.

8) To determine the relative order in which two records are returned from the file referenced by file-name-1, the contents of corresponding key data items are compared according to the rules for comparison of operands in a relation condition, starting with the most significant key data item.

a) If the contents of the corresponding key data items are not equal and the key is associated with the ASCENDING phrase, the record containing the key data item with the lower value is returned first;

b) If the contents of the corresponding key data items are not equal and the key is associated with the DESCENDING phrase, the record containing the key data item with the higher value is returned first; and,

c) If the contents of the corresponding key data items are equal, the determination is made on the contents of the next most significant key data item.

9) The execution of the SORT statement consists of three distinct phases as follows:

a) Records are made available to the file referenced by file-name-1. If INPUT PROCEDURE is specified, the execution of RELEASE statements in the input procedure makes the records available. If USING is specified, implicit READ and RELEASE statements make the records available. If the file referenced by file-name-2 is in an open mode when this phase commences, the EC-SORT-MERGE-FILE-OPEN exception condition is set to exist and the results of the execution of the SORT statement are undefined. When this phase terminates, the file referenced by file-name-2 is not in an open mode.

b) The file referenced by file-name-1 is sequenced. No processing of the files referenced by file-name-2 and file-name-3 takes place during this phase.

c) The records of the file referenced by file-name-1 are made available in sorted order. The sorted records are either written to the file referenced by file-name-3 or, by the execution of a RETURN statement, are made available for processing by the output procedure. If the file referenced by file-name-3 is in an open mode when this phase commences, the EC-SORT-MERGE-FILE-OPEN exception condition is set to exist and the results of the execution of the SORT statement are undefined. When this phase terminates, the file referenced by file-name-3 is not in the open mode.

10) The input procedure may consist of any procedure needed to create the records that are to be made available to the sort mechanism by executing RELEASE statements. The range includes all statements that are executed as the result of a transfer of control in the range of the input procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the input procedure. If the range of the input procedure causes the execution of any MERGE, RETURN, or format 1 SORT statements, the EC-SORT-MERGE-ACTIVE exception condition is set to exist and the results of the execution of the SORT statement are undefined. (See 14.6.3, Explicit and implicit transfers of control.)

11) If an input procedure is specified, control is passed to the input procedure before the file referenced by file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism after the last statement in the input procedure. When control passes to that return mechanism, the records that have been released to the file referenced by file-name-1 are sorted.

12) If the USING phrase is specified, all the records in the file(s) referenced by file-name-2 are transferred to the file referenced by file-name-1. For each of the files referenced by file-name-2 the execution of the SORT statement causes the following actions to be taken:

    a) The processing of the file is initiated. If the file-control entry for the file has a SHARING clause with the ALL phrase, the initiation is performed as if an OPEN statement with the INPUT phrase and the SHARING WITH READ ONLY phrase had been executed; otherwise, the initiation is performed as if an OPEN statement with the INPUT phrase and without a SHARING phrase is executed. The absence of the SHARING phrase means that the sharing mode is completely determined by the SHARING clause, if any, in the file control entry for the file connector referenced by file-name-2. If a nonfatal exception condition exists as a result of the execution of the implicit OPEN statement and there is an applicable USE procedure that completes normally or if there is no applicable USE procedure, the SORT statement continues as if the exception condition did not exist.

    b) The logical records are obtained and released to the sort operation. Each record is obtained as if a READ statement with the NEXT phrase, the IGNORING LOCK phrase, and the AT END phrase had been executed. When the at end condition exists for file-name-1, the processing for that file connector is terminated. If the file referenced by file-name-1 is described with variable-length records, the size of any record released to file-name-1 is the size of that record when it was read from file-name-2, regardless of the content of the data item referenced by the DEPENDING ON phrase of either a RECORD IS VARYING clause or an OCCURS clause specified in the sort-merge file description entry for file-name-1. If the size of the record read from the file referenced by file-name-2 is larger than the largest record allowed in the file description entry for file-name-1, the EC-SORT-MERGE-RELEASE exception condition is set to exist and the execution of the SORT statement is terminated. If file-name-1 is specified with variable-length records and the size of the record read from the file referenced by file-name-2 is smaller than the smallest record allowed in the file description entry for file-name-1, the EC-SORT-MERGE-RELEASE exception condition is set to exist and the execution of the SORT statement is terminated. If a fatal exception condition exists for file-name-1, the SORT is terminated.

    c) The processing of file-name-1 is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. This termination is performed before the file referenced by file-name-1 is sequenced by the SORT statement. For a relative file, the content of the relative key data item associated with file-name-2 is undefined after the execution of the SORT statement if file-name-2 is not referenced in the GIVING phrase.

    These implicit functions are performed such that any applicable USE procedures are executed; however, the execution of such a USE procedure shall not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, file-name-2.

    The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the file description entry for file-name-2 is undefined upon completion of the SORT statement.

13) The output procedure may consist of any procedure needed to process the records that are made available one at a time by the RETURN statement in sorted order from the file referenced by file-name-1. The range includes all statements that are executed as the result of a transfer of control in the range of the output procedure, as

well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure. If the range of the output procedure causes the execution of any MERGE, RELEASE, or format 1 SORT statement, the EC-SORT-MERGE-ACTIVE exception condition is set to exist and the results of the execution of the SORT statement are undefined. (See 14.6.3, Explicit and implicit transfers of control.)

14) If an output procedure is specified, control passes to it after the file referenced by file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism after the last statement in the output procedure. When control passes to that return mechanism, the mechanism provides for the termination of the sort and then passes control to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it selects the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

    NOTE This return mechanism transfers control from the end of the output procedure and is not associated with the RETURN statement.

15) If the GIVING phrase is specified, all the sorted records are written on the file referenced by file-name-3 as the implied output procedure for the SORT statement. For each of the files referenced by file-name-3, the execution of the SORT statement causes the following actions to be taken:

    a) The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT and SHARING WITH NO OTHER phrases had been executed. This initiation is performed after the execution of any input procedure.

    b) The sorted logical records are returned and written onto the file. Each record is written as if a WRITE statement without any optional phrases had been executed. If the file referenced by file-name-3 is described with variable-length records, the size of any record written to file-name-3 is the size of that record when it was read from file-name-1, regardless of the content of the data item referenced by the DEPENDING ON phrase of either a RECORD IS VARYING clause or an OCCURS clause specified in the file description entry for file-name-3.

       For a relative file, the relative key data item for the first record returned has the value 1; for the second record returned, the value 2; etc. After execution of the SORT statement, the content of the relative key data item indicates the last record returned to the file.

    c) The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

    These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed; however, the execution of such a USE procedure shall not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, file-name-3. On the first attempt to write beyond the externally defined boundaries of the file, any USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-3 is executed; if that USE procedure completes normally or if no such USE procedure is specified, the processing of the file is terminated as in general rule 15c above. If a fatal exception condition exists for file-name-3 as a result of the implicit OPEN during file initiation, the SORT is terminated. If a nonfatal exception condition exists for file-name-3 as a result of the implicit OPEN during file initiation, and there is an applicable USE procedure that completes normally, or no applicable USE procedure is available, the SORT statement continues.

    The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the sort-merge file description entry for file-name-1 is undefined upon completion of the SORT statement for which the GIVING phrase is specified.

16) If the file referenced by file-name-3 contains only fixed-length records, any record in the file referenced by file-name-1 containing fewer character positions than that fixed-length is space filled on the right to that fixed length, beginning with the first character position after the last character in the record, when that record is returned to the file referenced by file-name-3, as follows:

a) If there is only one record description entry associated with the file referenced by file-name-2 and that record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.

b) If there are multiple record description entries associated with the file referenced by file-name-2 and the descriptions include a SELECT WHEN clause, the rules of the SELECT WHEN clause are applied to the record to select its description. When the record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.

c) Otherwise, the record is space filled with alphanumeric space characters.

17) If a USE procedure invoked while a format 1 SORT statement is active does not complete normally, the SORT statement is terminated.

FORMAT 2

18) The SORT statement sorts the table referenced by data-name-2 and presents the sorted table in data-name-2 either in the order determined by the ASCENDING or DESCENDING phrases, if specified, or in the order determined by the KEY phrase associated with data-name-2.

19) To determine the relative order in which the table elements are stored after sorting, the contents of corresponding key data items are compared according to the rules for comparison of operands in a relation condition, starting with the most significant key data item.

a) If the contents of the corresponding key data items are not equal and the key is associated with the ASCENDING phrase, the table element containing the key data item with the lower value has the lower occurrence number.

b) If the contents of the corresponding key data items are not equal and the key is associated with the DESCENDING phrase, the table element containing the key data item with the higher value has the lower occurrence number.

c) If the contents of the corresponding key data items are equal, the determination is based on the contents of the next most significant key data item.

20) The number of occurrences of table elements referenced by data-name-2 is determined by the rules in the OCCURS clause.

21) If the KEY phrase is not specified, the sequence is determined by the KEY phrase in the data description entry of the table referenced by data-name-2.

22) If the KEY phrase is specified, it overrides any KEY phrase specified in the data description entry of the table referenced by data-name-2.

23) If data-name-1 is omitted, the data item referenced by data-name-2 is the key data item.

24) The sorted table elements of the table referenced by data-name-2 are placed in the table referenced by data-name-2.

14.9.37 START statement

The START statement provides a basis for logical positioning within a file, for subsequent sequential retrieval of records.

14.9.37.1 General format

```
START file-name-1
    [ FIRST
    | KEY relational-operator { data-name-1 | record-key-name-1 } [ WITH LENGTH arithmetic-expression-1 ]
    | LAST ]
[ INVALID KEY imperative-statement-1
| NOT INVALID KEY imperative-statement-2 ]
[ END-START ]
```

14.9.37.2 Syntax rules

1) The access mode of the file referenced by file-name-1 shall be either sequential or dynamic.

2) If the organization of the file referenced by file-name-1 is sequential, either the FIRST or the LAST phrase shall be specified.

3) In the KEY phrase, relational-operator is a relational operator specified in the general-relation format of 8.8.4.1.1, Relation conditions, with the exception of the relational operators 'IS NOT EQUAL TO' or 'IS NOT='.

4) Data-name-1 or record-key-name-1 may be qualified.

5) For relative files, data-name-1, if specified, shall be the data item specified in the RELATIVE KEY clause in the associated file control entry.

6) For indexed files, data-name-1, if specified, shall reference either:

   a) A data item specified as a prime or alternate record key associated with file-name-1, or

   b) A data item with the following characteristics:

      1. Its leftmost character position within a record of the file corresponds to the leftmost character position of a prime or alternate record key that is associated with file-name-1 and that is defined without the SOURCE phrase in the RECORD KEY clause or ALTERNATE RECORD KEY clause.

      2. It has the same class, category, and usage as that record key.

      3. Its length is not greater than the length of that record key.

7) Record-key-name-1 shall be specified with the SOURCE phrase in the RECORD KEY clause or in the ALTERNATE RECORD KEY clause in the file control entry for file-name-1.

8) If the LENGTH phrase is specified, file-name-1 shall reference a file with indexed organization.

14.9.37.3 General rules

1) The open mode of the file connector referenced by file-name-1 shall be input or I-O.

2) The execution of the START statement does not alter either the content of the record area or the content of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with file-name-1.

3) The execution of the START statement does not detect, acquire, or release record locks.

4) The execution of the START statement causes the value of the I-O status associated with file-name-1 to be updated. (See 9.1.12, I-O status.)

5) If, at the time of the execution of the START statement, the file position indicator indicates that an optional input file is not present, the invalid key condition exists and the execution of the START statement is unsuccessful.

6) Transfer of control following the successful or unsuccessful execution of the START operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the START statement as specified in 9.1.13, Invalid key condition.

7) Following the unsuccessful execution of a START statement, the file position indicator is set to indicate that no valid record position has been established. For indexed files, the key of reference is undefined.

RELATIVE FILES

8) If the KEY phrase is omitted, the START statement behaves as though KEY IS EQUAL TO data-name-1 had been specified, where data-name-1 is the name of the key specified in the RELATIVE KEY clause associated with file-name-1.

9) The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name-1 and a data item as specified in general rule 10. Numeric comparison rules apply. (See 8.8.4.1.1.3, Comparison of numeric operands.)

   a) If the relational operator is EQUAL, GREATER, NOT LESS, or GREATER OR EQUAL, the file position indicator is set to the relative record number of the first logical record in the file whose key satisfies the comparison searching the file sequentially.

   b) If the relational operator is LESS, NOT GREATER, or LESS OR EQUAL, the file position indicator is set to the relative record number of the first logical record in the file whose key satisfies the comparison searching the file in reverse order.

   c) If the comparison is not satisfied by any record in the file, the invalid key condition exists and the execution of the START statement is unsuccessful.

10) The comparison described in general rule 9 uses the data item referenced by the RELATIVE KEY clause in the file control entry associated with file-name-1.

11) If FIRST is specified, the file position indicator is set to the relative record number of the first existing logical record in the file. If no records exist in the file, the invalid key condition exists and the execution of the START statement is unsuccessful.

12) If LAST is specified, the file position indicator is set to the relative record number of the last existing logical record in the file. If no records exist in the file, the invalid key condition exists and the execution of the START statement is unsuccessful.

INDEXED FILES

13) The value of arithmetic-expression-1 reflects the number of characters that will be used as a partial key for positioning to a record in file-name-1. If data-name-1 or record-key-name-1 is of class alphanumeric, arithmetic-expression-1 is the number of alphanumeric character positions; if data-name-1 or record-key-name-1 is of class national, arithmetic-expression-1 is the number of national character positions.

14) If arithmetic-expression-1 does not evaluate to a positive nonzero integer that is less than or equal to the length of the associated key, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.

15) If the KEY phrase is not specified, the behavior is the same as if KEY IS EQUAL TO data-name-1 or record-key-name-1 had been specified, with data-name-1 or record-key-name-1 being the prime record key for the file.

16) The key specified in the KEY phrase, or that shares a leftmost character with the data item specified in the KEY phrase, becomes the key of reference. This key of reference is used to establish the ordering of records for the purpose of this START statement as indicated in general rule 17. If the execution of the START statement is successful, this key of reference is used for subsequent sequential READ statements referencing file-name-1.

17) Execution of the START statement behaves as if:

    a) The specified key is set up by moving the relevant parts of the record area into a temporary data area.

    b) The length of this temporary area is considered to be the length specified in the LENGTH clause, if specified, or else the length of record-key-name-1, if specified, or else the length of data-name-1.

    c) If the relational operator is EQUAL, GREATER, NOT LESS, or GREATER OR EQUAL, the file is searched sequentially with the key of reference being extracted from each record in turn into another temporary area. This second temporary area is truncated to the same length as the first.

    d) If the relational operator is LESS, NOT GREATER, or LESS OR EQUAL, the file is searched in reverse order with the key of reference being extracted from each record in turn into another temporary area. This second temporary area is truncated to the same length as the first.

    e) The comparison specified by the relational operator in the KEY phrase is made between these two temporary areas, with the second temporary area on the left hand side, and according to the collating sequence of the file. Comparison proceeds as specified for items of the class of data-name-1 and operands of equal length in 8.8.4.1.1.6, Comparison of alphanumeric operands, or 8.8.4.1.1.8, Comparison of national operands. Then, either:

        1. The file position indicator is set to the value of the key of reference in the first logical record whose key satisfies the comparison, or

        2. If the comparison is not satisfied by any record in the file, the invalid key condition exists and the execution of the START statement is unsuccessful.

18) If FIRST is specified, the file position indicator is set to the value of the primary key of the first existing logical record in the physical file and the key of reference is set to the primary key. If no records exist in the file, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.

19) If LAST is specified, the file position indicator is set to the value of the primary key of the last existing logical record in the physical file and the key of reference is set to the primary key. If no records exist in the file, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.

SEQUENTIAL FILES

20) If FIRST is specified, the file position indicator is set to 1 if records exist in the physical file. If no records exist in the file, or the physical file does not support the ability to position at the first record, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.

21) If LAST is specified, the file position indicator is set to the record number of the last existing logical record in the physical file. If no records exist in the file, or the physical file does not support the ability to position at the last record, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.

14.9.38 STOP statement

The STOP statement causes termination of the execution of the run unit.

14.9.38.1 General format

14.9.38.2 Syntax rules

1) If a STOP statement appears in a consecutive sequence of imperative statements within a sentence, it shall appear as the last statement in that sequence.

2) Identifier-1 shall reference an integer data item or a data item with usage display or usage national.

3) If literal-1 is numeric, it shall be an integer.

4) Literal-1 shall not be a zero-length literal.

14.9.38.3 General rules

1) The operations described in 14.6.10, Normal run unit termination, are performed.

2) If the ERROR phrase is specified, the operating system will indicate an error termination of the run unit if such a capability exists within the operating system.

3) If the NORMAL phrase is specified, the operating system will indicate a normal termination of the run unit if such a capability exists within the operating system.

4) If neither the ERROR phrase nor the NORMAL phrase is specified, the operating system will indicate a normal termination of the run unit if such a capability exists within the operating system unless error termination has been indicated by an implementor-defined mechanism.

5) During execution of the STOP statement with literal-1 or identifier-1 specified, literal-1 or the contents of the data item referenced by identifier-1 are passed to the operating system. Any constraints on the value of literal-1 or the contents of the data item referenced by identifier-1 are defined by the implementor.

6) Execution of the run unit terminates and control is transferred to the operating system.

14.9.39 STRING statement

The STRING statement provides concatenation of the partial or complete contents of one or more data items into a single data item.

14.9.39.1 General format

```
STRING { { identifier-1 } ... [ DELIMITED BY { identifier-2 } ] } ...
       { { literal-1    }                    { literal-2    } }
                                             { SIZE         }
    INTO identifier-3
    [ WITH POINTER identifier-4 ]
    [ ON OVERFLOW imperative-statement-1 ]
    [ NOT ON OVERFLOW imperative-statement-2 ]
    [ END-STRING ]
```

14.9.39.2 Syntax rules

1) All literals shall be described as alphanumeric, boolean, or national literals, and all identifiers, except identifier-4, shall be described implicitly or explicitly as usage display or national. If any one of literal-1, literal-2, identifier-1, identifier-2, or identifier-3 is of class national, then all shall be of class national.

2) Literal-1 or literal-2 shall not be a figurative constant that begins with the word ALL.

3) Literal-2 shall not be a zero-length literal.

4) Identifier-3 shall not be reference modified.

5) Identifier-3 shall not reference an edited data item and shall not be described with the JUSTIFIED clause.

6) Identifier-3 shall not reference a strongly-typed group item.

7) Identifier-4 shall be described as an elementary numeric integer data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by identifier-3. The symbol 'P' shall not be used in the picture character-string of identifier-4.

8) Where identifier-1 or identifier-2 is an elementary numeric data item, it shall be described as an integer without the symbol 'P' in its picture character-string.

9) The DELIMITED phrase may be omitted only immediately preceding the INTO phrase. If it is omitted, DELIMITED BY SIZE is implied.

10) Literal-1 or the data item referenced by identifier-1 is the sending operand. The data item referenced by identifier-3 is the receiving operand.

11) Identifier-1, identifier-2, or identifier-3 shall not specify a variable-length group.

14.9.39.3 General rules

1) Literal-2 or the content of the data item referenced by identifier-2 indicates the character(s) delimiting the move. If the SIZE phrase is used, the content of the complete data item defined by identifier-1 or literal-1 is

moved. If the SIZE phrase is associated with an identifier-1 that references an any-length elementary item, the current length of the data item is used to determine the number of characters moved.

2) When a figurative constant is specified as literal-1 or literal-2, it refers to an implicit one character data item whose usage shall be the same as the usage of identifier-3, either display or national.

3) When the STRING statement is executed, the transfer of data is governed by the following rules:

   a) Characters from literal-1 or from the content of the data item referenced by identifier-1 are transferred to the data item referenced by identifier-3 in accordance with the MOVE statement rules for alphanumeric-to-alphanumeric moves or, when identifier-3 is of class national, national-to-national moves, except that no space filling is provided. If an identifier-1 references a zero-length item, that sending operand is ignored.

   b) If the DELIMITED phrase is specified and literal-2 is specified or identifier-2 is specified and is not a zero-length item, the content of the data item referenced by identifier-1, or the value of literal-1, is transferred to the receiving data item in the sequence specified in the STRING statement beginning with the leftmost character positions and continuing from left to right until the end of the sending data item is reached or the end of the receiving data item is reached or until the character(s) specified by literal-2, or by the content of the data item referenced by identifier-2, are encountered. The character(s) specified by literal-2 or by the data item referenced by identifier-2 are not transferred.

   c) If the DELIMITED phrase is specified and the SIZE phrase is specified or identifier-2 is specified and is a zero-length item, the entire content of literal-1, or the content of the data item referenced by identifier-1, is transferred, in the sequence specified in the STRING statement, to the data item referenced by identifier-3 until all data has been transferred or the end of the data item referenced by identifier-3 has been reached.

   This behavior is repeated until all occurrences of literal-1 or data items referenced by identifier-1 have been processed.

4) If the POINTER phrase is specified, the data item referenced by identifier-4 shall have a value greater than zero at the start of execution of the STRING statement.

5) If the POINTER phrase is not specified, the following general rules apply as if the user had specified identifier-4 referencing a data item with an initial value of 1.

6) When characters are transferred to the data item referenced by identifier-3, the moves behave as though the characters were moved one at a time from the source into the character positions of the data item referenced by identifier-3 designated by the value of the data item referenced by identifier-4 (provided the value of the data item referenced by identifier-4 does not exceed the length of the data item referenced by identifier-3), and then the data item referenced by identifier-4 was increased by one prior to the move of the next character or prior to the end of execution of the STRING statement. The value of the data item referenced by identifier-4 is changed during execution of the STRING statement only by the behavior specified above.

7) At the end of execution of the STRING statement, only the portion of the data item referenced by identifier-3 that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by identifier-3 will contain data that was present before this execution of the STRING statement.

8) Before each move of a character to the data item referenced by identifier-3, if the value associated with the data item referenced by identifier-4 is either less than one or exceeds the number of character positions in the data item referenced by identifier-3, the following occurs:

   a) No further data is transferred to the data item referenced by identifier-3.

   b) The EC-OVERFLOW-STRING exception condition is set to exist.

c) If the ON OVERFLOW phrase is specified, control is transferred to imperative-statement-1 and execution continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the STRING statement.

d) If the ON OVERFLOW phrase is not specified, execution continues as specified in 14.6.12.1.3, Non fatal exception conditions.

e) The NOT ON OVERFLOW phrase, if specified, is ignored.

9) If, at the time of execution of a STRING statement with the NOT ON OVERFLOW phrase, the conditions described in general rule 8 are not encountered, after completion of the transfer of data according to the other general rules, the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the STRING statement or, if the NOT ON OVERFLOW phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the STRING statement.

10) If identifier-1, or identifier-2, occupies the same storage area as identifier-3, or identifier-4, or if identifier-3 and identifier-4 occupy the same storage area, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See 14.6.9, Overlapping operands.)

14.9.40 SUBTRACT statement

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more items equal to the results.

14.9.40.1 General format

Format 1 (simple):

```
SUBTRACT { identifier-1 } ... FROM { identifier-2 [ rounded-phrase ] } ...
         { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-SUBTRACT ]
```

Format 2 (giving):

```
SUBTRACT { identifier-1 } ... FROM { identifier-2 }
         { literal-1    }         { literal-2    }
    GIVING { identifier-3 [ rounded-phrase ] } ...
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-SUBTRACT ]
```

Format 3 (corresponding):

```
SUBTRACT { CORRESPONDING } identifier-4 FROM identifier-5 [ rounded-phrase ]
         { CORR          }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-SUBTRACT ]
```

where rounded-phrase is described in 14.7.3, ROUNDED phrase.

14.9.40.2 Syntax rules

ALL FORMATS

1) When native arithmetic is in effect, the composite of operands described in 14.7.6, Arithmetic statements, is determined as follows:

   a) In format 1, by using all of the operands in the statement.

   b) In format 2, by using all of the operands in the statement excluding the data item that follow the word GIVING.

c) In format 3, by using the two corresponding operands for each separate pair of corresponding items.

FORMATS 1 AND 2

2) Identifier-1 and identifier-2 shall reference numeric data items.

3) Literal-1 and literal-2 shall be numeric literals.

FORMAT 2

4) Identifier-3 shall reference a numeric data item or a numeric-edited data item.

FORMAT 3

5) The words CORR and CORRESPONDING are equivalent.

6) Identifier-4 and identifier-5 shall be alphanumeric group items, national group items, variable-length groups, or strongly-typed group items and shall not be described with level-number 66.

14.9.40.3 General rules

1) When format 1 is used, the initial evaluation consists of determining the value to be subtracted, which is literal-1 or the value of the data item referenced by identifier-1, or if more than one is specified, the sum of such operands. The initial evaluation is subtracted from the value of the data item referenced by identifier-2 and the result is stored as the new value of the data item referenced by identifier-2.

When standard arithmetic, standard-decimal arithmetic, or standard-binary arithmetic is in effect, the result of the initial evaluation is equivalent to the result of the arithmetic expression

$$(\text{operand-}1_1 + \text{operand-}1_2 + \ldots + \text{operand-}1_n)$$

where the values of operand-1 are the values of literal-1 and the data items referenced by identifier-1 in the order in which they are specified in the SUBTRACT statement. The result of the subtraction from the value of each data item referenced by identifier-2 is equivalent to the result of the arithmetic expression

(identifier-2 – initial-evaluation)

where initial-evaluation represents the result of the initial evaluation.

2) When format 2 is used, the initial evaluation consists of determining the value to be subtracted, which is literal-1 or the value of the data item referenced by identifier-1, or if more than one is specified, the sum of such operands; and subtracting this value from literal-2 or the value of the data item referenced by identifier-2. The result is stored as the new value of the data item referenced by identifier-3.

When standard arithmetic, standard-decimal arithmetic, or standard-binary arithmetic is in effect, the result of the arithmetic expression

$$(\text{operand-}2 - (\text{operand-}1_1 + \text{operand-}1_2 + \ldots + \text{operand-}1_n))$$

where the values of operand-1 are the values of literal-1 and the data items referenced by identifier-1 in the order in which they are specified in the SUBTRACT statement and the value of operand-2 is the value of either literal-2 or the data item referenced by identifier-2 in the SUBTRACT statement.

3) When format 3 is used, data items in identifier-4 are subtracted from and stored in corresponding items in identifier-5.

When standard arithmetic, standard-decimal arithmetic, or standard-binary arithmetic is in effect, the result of the subtraction is equivalent to

(operand-1 – operand-2)

where the value of operand-1 is the value of the data item in identifier-4 and the value of operand-2 is the value of the corresponding data item in identifier-5.

4) When native arithmetic is in effect and none of the operands is described with usage binary-char, binary-short, binary-long, binary-double, float-short, float-long, or float-extended, enough places shall be carried so as not to lose significant digits during execution.

5) Data items within identifier-4 are selected to be subtracted from selected data items within identifier-5 according to the rules specified in 14.7.5, CORRESPONDING phrase. The results are the same as if the user had referred to each pair of corresponding identifiers in separate SUBTRACT statements.

6) Additional rules and explanations relative to this statement are given in 14.6.12.2, Incompatible data; 14.7.3, ROUNDED phrase; 14.7.4, SIZE ERROR phrase and size error condition; 14.7.5, CORRESPONDING phrase; and 14.7.6, Arithmetic statements.

14.9.41 SUPPRESS statement

The SUPPRESS statement inhibits the printing of a report group.

14.9.41.1 General format

SUPPRESS PRINTING

14.9.41.2 Syntax rules

1) The SUPPRESS statement may appear only in a USE BEFORE REPORTING procedure.

14.9.41.3 General rules

1) The SUPPRESS statement inhibits printing only for the report group named in the USE procedure within which the SUPPRESS statement appears.

2) The effect of the SUPPRESS statement is limited to the current instance of the report group.

   NOTE A SUPPRESS statement should be executed again on each further occasion that the associated report group is to be inhibited.

3) When the SUPPRESS statement is executed, the following report group functions are inhibited:

   a) The printing of the print lines of the report group.

   b) Any page advance associated with the report group.

   c) The processing of any NEXT GROUP clause in the report group.

   d) Any changes to LINE-COUNTER associated with the report group.

   e) If the associated report group is a detail, the SUPPRESS statement does not affect the sensing for control breaks or the subsequent control break processing.

14.9.42 TERMINATE statement

The TERMINATE statement completes the processing of the specified reports.

14.9.42.1 General format

TERMINATE { report-name-1 } ...

14.9.42.2 Syntax rules

1) Report-name-1 shall be defined by a report description entry in the report section.

2) If report-name-1 is defined in a containing program, the file description entry associated with report-name-1 shall contain a GLOBAL clause.

14.9.42.3 General rules

1) The TERMINATE statement may be executed only for a report that is in the active state. If the report is not in the active state, the EC-REPORT-INACTIVE exception condition is set to exist and the execution of the statement has no other effect.

2) If no GENERATE statement has been executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement for that report, the TERMINATE statement causes no processing of any kind to take place for any report groups and has the sole effect of changing the state of the report to inactive.

3) If at least one GENERATE statement has been executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement for that report, the TERMINATE statement causes the following actions to take place:

   a) The contents of any control data items are changed to their prior values.

   b) Each control footing is printed, if defined, beginning with the minor control footing, as defined for the GENERATE statement, as though a control break has been sensed in the most major control data item.

   c) The report footing is printed, if defined.

   d) The contents of any control data items are restored to the values they had at the start of execution of the TERMINATE statement.

4) The result of executing a TERMINATE statement in which more than one report-name-1 is specified is as though a separate TERMINATE statement had been executed for each report-name-1 in the same order as specified in the statement. If an implicit TERMINATE statement results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, processing resumes at the next implicit TERMINATE statement, if any.

5) If a nonfatal exception condition is raised during the execution of a TERMINATE statement, execution resumes at the next report item, line, or report group, whichever follows in logical order.

6) The TERMINATE statement does not close the file associated with report-name-1.

14.9.43 UNLOCK statement

The UNLOCK statement explicitly releases any record locks associated with a file connector.

14.9.43.1 General format

UNLOCK file-name-1 [RECORD | RECORDS]

14.9.43.2 Syntax rules

1) File-name-1 shall not refer to a sort file or a merge file.

14.9.43.3 General rules

1) Any record locks associated with the file connector referenced by file-name-1 are released by the execution of the UNLOCK statement. The presence or absence of any record locks does not affect the success of the execution of the UNLOCK statement.

   NOTE To unlock one particular record, use the READ statement with the NO LOCK phrase.

2) File-name-1 shall reference a file connector in the open mode.

3) The execution of the UNLOCK statement causes the value of the I-O status of the file connector referenced by file-name-1 to be updated.

14.9.44 UNSTRING statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

14.9.44.1 General format

```
UNSTRING identifier-1
    [ DELIMITED BY [ ALL ] { identifier-2 | literal-1 } [ OR [ ALL ] { identifier-3 | literal-2 } ] ... ]
    INTO { identifier-4 [ DELIMITER IN identifier-5 ] [ COUNT IN identifier-6 ] } ...
    [ WITH POINTER identifier-7 ]
    [ TALLYING IN identifier-8 ]
    [ ON OVERFLOW imperative-statement-1 | NOT ON OVERFLOW imperative-statement-2 ]
    [ END-UNSTRING ]
```

14.9.44.2 Syntax rules

1) Literal-1 and literal-2 shall be literals of the category alphanumeric or national and shall be neither a figurative constant that begins with the word ALL nor a zero-length literal.

2) Identifier-1, identifier-2, identifier-3, and identifier-5 shall reference data items of category alphanumeric or national.

3) If any of identifier-1, identifier-2, identifier-3, identifier-4, identifier-5, literal-1, or literal-2 are of category national, then all shall be of category national.

4) Identifier-4 shall be described implicitly or explicitly as usage display and category alphabetic, alphanumeric, or numeric; or as usage national and category national or numeric. Numeric items shall not be specified with the symbol 'P' in their picture character-string.

5) Identifier-6 and identifier-8 shall reference integer data items. The symbol 'P' shall not be used in the picture character-string.

6) Identifier-7 shall be described as an elementary numeric integer data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by identifier-1. The symbol 'P' shall not be used in the picture character-string of identifier-7.

7) The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

8) The data item referenced by identifier-1 is the sending operand.

9) The data item referenced by identifier-4 is the receiving operand for data. The data item referenced by identifier-5 is the receiving operand for delimiters.

10) Identifier-1, identifier-2, identifier-3, identifier-4 or identifier-5, shall not reference a variable-length group.

14.9.44.3 General rules

1) All references to identifier-2 and literal-1 apply equally to identifier-3 and literal-2, respectively, and all recursions thereof.

2) If the data item referenced by identifier-1 is a zero-length item, execution of the UNSTRING statement terminates immediately.

3) Literal-1 or the data item referenced by identifier-2 specifies a delimiter.

4) The data item referenced by identifier-6 represents the count of the number of characters within the data item referenced by identifier-1 isolated by the delimiters for the move to the data item referenced by identifier-4. This value does not include a count of the delimiter character(s).

5) The data item referenced by identifier-7 contains a value that indicates a relative character position within the area referenced by identifier-1.

6) The data item referenced by identifier-8 is a counter that is incremented by 1 for each occurrence of the data item referenced by identifier-4 accessed during the UNSTRING operation.

7) When a figurative constant is used as the delimiter, it stands for a single-character national literal if identifier-1 is a national data item; otherwise, it stands for a single-character alphanumeric literal.

   When the ALL phrase is specified, one occurrence or two or more contiguous occurrences of literal-1 (figurative constant or not) or the content of the data item referenced by identifier-2 are treated as if they were only one occurrence, and one occurrence of literal-1 or the data item referenced by identifier-2 is moved to the receiving data item according to the rules in general rule 11d.

8) When any examination encounters two contiguous delimiters, the current receiving area shall be space-filled if it is described as alphabetic, alphanumeric, or national; or zero-filled if it is described as numeric.

9) Each literal-1 or the data item referenced by identifier-2 represents one delimiter. When a delimiter contains two or more characters, all of the characters shall be present in contiguous positions of the sending item, and in the order given, to be recognized as a delimiter. If the data item referenced by identifier-2 or identifier-3 is a zero-length item, that delimiter is ignored. When neither literal-1 nor literal-2 is specified and all data items referenced by identifier-2 and identifier-3 are zero-length items, it is as if the DELIMITED phrase were not specified.

10) When two or more delimiters are specified in the DELIMITED BY phrase, an OR condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field shall be considered a part of more than one delimiter.

    Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

11) When the UNSTRING statement is initiated, the current receiving area is the data item referenced by identifier-4. Data is transferred from the data item referenced by identifier-1 to the data item referenced by identifier-4 according to the following rules:

    a) If the POINTER phrase is specified, the string of characters referenced by identifier-1 is examined beginning with the relative character position indicated by the content of the data item referenced by identifier-7. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.

    b) If the DELIMITED BY phrase is specified, the examination proceeds left to right until a delimiter specified by either literal-1 or the value of the data item referenced by identifier-2 is encountered. (See general rule 9.) If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate

character position, the number of characters examined is one less than the size of the current receiving area. Size is defined as number of character positions.

If the end of the data item referenced by identifier-1 is encountered before the delimiting condition is met, the examination terminates with the last character examined.

c) The characters examined, excluding any delimiting characters, shall be treated as an elementary national data item if identifier-1 is of category national, and otherwise as an elementary alphanumeric data item, and shall be moved into the current receiving area according to the rules for the MOVE statement.

d) If the DELIMITER IN phrase is specified the delimiting character(s) shall be treated as an elementary national data item if identifier-1 is of category national, and otherwise as an elementary alphanumeric data item and shall be moved into the data item referenced by identifier-5 according to the rules for the MOVE statement. If the delimiting condition is the end of the data item referenced by identifier-1, then the data item referenced by identifier-5 is space filled.

e) If the COUNT IN phrase is specified, a value equal to the number of characters examined, excluding any delimiter characters, shall be moved into the area referenced by identifier-6 according to the rules for an elementary move.

f) If the DELIMITED BY phrase is specified the string of characters is further examined beginning with the first character position to the right of the delimiter. If the DELIMITED BY phrase is not specified the string of characters is further examined beginning with the character position to the right of the last character transferred.

g) After data is transferred to the data item referenced by identifier-4, the current receiving area is the data item referenced by the next recurrence of identifier-4. The behavior described in general rules 12b through 12f is repeated until either all the characters are exhausted in the data item referenced by identifier-1, or until there are no more receiving areas.

12) The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

13) The content of the data item referenced by identifier-7 will be incremented by one for each character examined in the data item referenced by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is completed, the content of the data item referenced by identifier-7 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

14) When the execution of an UNSTRING statement with a TALLYING phrase is completed, the content of the data item referenced by identifier-8 contains a value equal to its value at the beginning of the execution of the statement plus a value equal to the number of identifier-4 receiving data items accessed during execution of the statement.

15) Either of the following situations causes an overflow condition:

a) An UNSTRING is initiated, and the value in the data item referenced by identifier-7 is less than 1 or greater than the number of character positions described for the data item referenced by identifier-1.

b) If, during execution of an UNSTRING statement, all receiving areas have been acted upon, and the data item referenced by identifier-1 contains characters that have not been examined.

16) When an overflow condition exists, the following occurs:

a) The UNSTRING operation is terminated.

b) The EC-OVERFLOW-UNSTRING exception condition is set to exist.

c) If the ON OVERFLOW phrase is specified, control is transferred to imperative-statement-1 and execution continues according to the rules for each statement in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement: otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the UNSTRING statement.

d) If the ON OVERFLOW phrase is not specified, execution continues as specified in 14.6.12.1.3, Non fatal exception conditions

e) The NOT ON OVERFLOW phrase, if specified, is ignored.

17) If, at the time of execution of an UNSTRING statement, the conditions described in general rule 15 are not encountered, after completion of the transfer of data according to the other general rules, the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the UNSTRING statement or, if the NOT ON OVERFLOW phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the UNSTRING statement.

18) If identifier-1, identifier-2, or identifier-3, occupies the same storage area as identifier-4, identifier-5, identifier-6, identifier-7, or identifier-8, or if identifier-4, identifier-5, or identifier-6, occupies the same storage area as identifier-7 or identifier-8, or if identifier-7 and identifier-8 occupy the same storage area, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See 14.6.9, Overlapping operands.)

14.9.45 USE statement

The USE statement specifies:

— Procedures for error or exception handling that are in addition to the standard procedures provided by other facilities,

NOTE These facilities can include the input-output control system and the operating system.

— A procedure to be executed just before the printing of the designated report group, or

— Procedures that are executed after the detection of exception conditions.

14.9.45.1 General format

Format 1 (file-exception):

```
USE [ GLOBAL ] AFTER STANDARD { EXCEPTION } PROCEDURE ON { { file-name-1 } ... }
                              { ERROR     }               { INPUT             }
                                                          { OUTPUT            }
                                                          { I-O               }
                                                          { EXTEND            }
```

Format 2 (reporting):

```
USE [ GLOBAL ] BEFORE REPORTING identifier-1
```

Format 3 (exception-name):

```
USE AFTER { EXCEPTION CONDITION } { exception-name-1                    }
          { EC                   } { exception-name-2 { FILE file-name-2 } ... } ...
```

Format 4 (exception-object):

```
USE AFTER { EXCEPTION OBJECT } { class-name-1     }
          { EO               } { interface-name-1 }
```

14.9.45.2 Syntax rules

ALL FORMATS

1) A USE statement, when present, shall immediately follow a section header in the declaratives portion of the procedure division and shall appear in a sentence by itself. The remainder of the section shall consist of zero, one, or more procedural paragraphs that define the procedures to be used.

2) File-name-1 or file-name-2 shall not be a sort or a merge file.

3) Within a declarative procedure, there shall be no reference to any nondeclarative procedures except in a RESUME statement.

4) Procedure-names within a declarative section may be referenced in a different declarative section or in a nondeclarative procedure only with a PERFORM statement.

FORMATS 1 AND 3

5) The files implicitly or explicitly referenced in the USE statement need not all have the same organization or access.

FORMAT 1

6) The words ERROR and EXCEPTION are synonymous and may be used interchangeably.

7) The INPUT, OUTPUT, I-O, and EXTEND phrases may each be specified only once in the declaratives portion of a given procedure division.

8) The same file-name shall not appear in more than one USE AFTER EXCEPTION statement within the same procedure division.

FORMAT 2

9) Identifier-1 shall reference a report group. The same identifier-1 shall not appear in more than one USE BEFORE REPORTING statement within the same procedure division.

10) The GENERATE, INITIATE, or TERMINATE statements shall not appear in a paragraph within a USE BEFORE REPORTING procedure.

11) A USE BEFORE REPORTING procedure shall not alter the value of any control data item.

FORMAT 3

12) EC is synonymous with EXCEPTION CONDITION.

13) Exception-name-2 shall be an exception-name beginning with 'EC-I-O'.

14) The same pair of exception-name-2 and file-name-2 shall not be specified in more than one USE statement within the same procedure division.

FORMAT 4

15) EO is synonymous with EXCEPTION OBJECT.

16) Class-name-1 shall be the name of a class specified in the REPOSITORY paragraph.

17) Interface-name-1 shall be the name of an interface specified in the REPOSITORY paragraph.

14.9.45.3 General rules

ALL FORMATS

1) The USE statement is never executed; it merely defines the conditions calling for the execution of the USE procedures.

2) During the execution of a USE procedure, if a statement raises an exception condition that would cause the execution of a USE procedure that had previously been activated and had not yet returned control to the activating entity, the EC-FLOW-USE exception condition is set to exist.

FORMATS 1, 3, AND 4

3) A declarative is selected for execution by analyzing the USE statements in the source element in the order in which they are specified. The first declarative that satisfies the selection criteria is executed and no other

declaratives are executed. If an exception object was raised, the method specified in general rule 13 is applied. Otherwise, the following rules are applied in order:

a) All format 1 USE statements in which file-name-1 is specified are examined using the criteria specified in general rule 6. If no qualifying USE statement is found, the USE statements in the source element are examined again.

b) All format 1 USE statements in which file-name-1 is not specified are examined using the criteria specified in general rule 6. If no qualifying USE statement is found, the USE statements in the source element are examined again.

c) All format 3 USE statements in which file-name-2 is specified and exception-name-2 is a level-3 exception-name are examined. If the exception condition that was raised matches exception-name-2 and the exception condition is associated with file-name-2, that declarative is executed. If no qualifying USE statement is found, the USE statements in the source element are examined again.

d) All format 3 USE statements in which file-name-2 is specified and exception-name-2 is a level-2 exception-name are examined. If the exception condition that was raised matches exception-name-2 and the exception condition is associated with file-name-2, that declarative is executed. If no qualifying USE statement is found, the USE statements in the source element are examined again.

e) All format 3 USE statements in which file-name-2 is not specified and exception-name-1 is a level-3 exception-name are examined. If the exception condition that was raised matches exception-name-1, that declarative is executed. If no qualifying USE statement is found, the USE statements in the source element are examined again.

f) All format 3 USE statements in which file-name-2 is not specified and exception-name-1 is a level-2 exception-name are examined. If the exception condition that was raised matches exception-name-1, that declarative is executed. If no qualifying USE statement is found, the USE statements in the source element are examined again.

g) Any format 3 USE statements in which file-name-2 is not specified and exception-name-1 is a level-1 exception-name are examined. If the exception condition that was raised matches exception-name-1, that declarative is executed. If no qualifying USE statement is found, and a containing source element contains a USE statement with the GLOBAL clause, the search is repeated as specified in general rule 4. If no declarative is identified, no declarative is executed.

FORMATS 1 AND 2

4) For source elements contained within other source elements, multiple declaratives may be eligible for selection for a given exception condition. The declarative selected for execution is determined in the following order of precedence:

   a) the qualifying declarative in the source element that contains the statement that caused the condition to exist,

   b) a qualifying declarative with the GLOBAL attribute in the next inclusive directly containing source element. This step is repeated with the next higher directly containing source element until a declarative is selected or the outermost source element is reached.

FORMAT 1

5) Within a given procedure division, a USE statement specifying file-name-1 takes precedence over any USE statements specifying an INPUT, OUTPUT, I-O, or EXTEND phrase.

6) The procedures associated with a USE statement are executed by the input-output control system after completion of the standard input-output exception routine upon the unsuccessful execution of an input-output

operation unless an AT END or INVALID KEY phrase takes precedence. The rules concerning when the procedures are executed are as follows:

a) If file-name-1 is specified, the associated procedure is executed when the condition described in the USE statement occurs.

b) If INPUT is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the input mode or in the process of being opened in the input mode, except those files referenced by file-name-1 in another USE statement specifying the same condition.

c) If OUTPUT is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the output mode or in the process of being opened in the output mode, except those files referenced by file-name-1 in another USE statement specifying the same condition.

d) If I-O is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the I-O mode or in the process of being opened in the I-O mode, except those files referenced by file-name-1 in another USE statement specifying the same condition.

e) If EXTEND is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the extend mode or in the process of being opened in the extend mode, except those files referenced by file-name-1 in another USE statement specifying the same condition.

7) After execution of the USE procedure, control is transferred to the invoking routine in the input-output control system. If the I-O status value does not indicate a fatal EC-I-O exception condition, the input-output control system returns control to the next executable statement following the input-output statement whose execution caused the exception. If the I-O status value does indicate a fatal EC-I-O exception condition, the implementor determines what action is taken as described in 9.1.12, I-O status.

FORMAT 2

8) The declarative is invoked just before the named report group is produced during the execution of the runtime element. The report group is named by identifier-1 in the USE BEFORE REPORTING statement that prefaces the declarative procedure.

9) The declarative procedure associated with the USE BEFORE REPORTING statement is implicitly performed on each occasion that the named report group is processed at the following logical point:

a) After any control break processing, if the associated report group is a detail and the associated report description has a CONTROL clause. (See 13.18.16, CONTROL clause.)

b) After any adding that is required to increment sum counters defined in the report group;

c) Before any page fit processing, if the associated report group is a body group and the report is divided into pages.

d) Before the processing of any LINE clauses defined for the report group and any printable items described in entries subordinate to them.

10) If a GENERATE, INITIATE, or TERMINATE statement is executed within the range of a declarative procedure whose USE statement contains the BEFORE REPORTING phrase, the EC-FLOW-REPORT exception condition is set to exist, the result of the execution of the GENERATE, INITIATE, or TERMINATE statement is unsuccessful, and the state of the report is unchanged.

FORMAT 3

11) The exception condition that is used to determine which USE statement to select is the first one that occurs in the order of evaluation of the statement that causes the exception to exist.

12) When exception-name-1 or exception-name-2 begins with EC-I-O, after execution of the USE procedure, control is transferred to the invoking routine in the input-output control system. If the I-O status value does not indicate a fatal input-output error, the input-output control system returns control to the next executable statement following the input-output statement whose execution caused the exception. If the I-O status value does indicate a fatal error, the implementor determines what action is taken. (See 9.1.12, I-O status.)

FORMAT 4

13) A declarative is selected for execution by analyzing the USE statements in a source element in the order in which they are specified. Zero or one declarative is selected for execution by applying the following rules in order:

    a) If class-name-1 is specified and the exception object that was raised is a factory object or instance object of class-name-1 or of a subclass of class-name-1, the associated declarative is executed and no other declaratives are executed; otherwise, all of the USE statements in the source element are analyzed again and:

    b) If interface-name-1 is specified and the exception object that was raised is described with an IMPLEMENTS clause that references interface-name-1, the associated declarative is executed and no other declaratives are executed; otherwise execution proceeds as specified in 14.6.12.1.4, Exception objects.

14) Upon entry to the associated declarative, the predefined object reference EXCEPTION-OBJECT references the exception object.

14.9.46 VALIDATE statement

The VALIDATE statement invokes data validation, input distribution, and error indication for a data item.

14.9.46.1 General format

VALIDATE { identifier-1 } ...

14.9.46.2 Syntax rules

1) Identifier-1 shall reference a data item described in the file, linkage, local-storage, or working-storage section. Identifier-1 shall not be reference modified.

2) Identifier-1 shall not reference a data item of class index, object, or pointer.

3) The data description entry for the data item referenced by identifier-1 or any data item subordinate to identifier-1 shall not contain a VALIDATE-STATUS clause referencing identifier-1 or an item subordinate to identifier-1.

4) Identifier-1 shall not reference a 66-level entry.

14.9.46.3 General rules

1) If identifier-1 or a group subordinate to identifier-1 is described with a GROUP-USAGE clause, format validation is performed on each elementary item within the bit group or national group. Any DEFAULT, DESTINATION, or VALUE clause specified, applies to the elementary item or group item on which it is specified, even when that item is explicitly or implicitly described with a GROUP-USAGE clause.

2) The data item specified by identifier-1 and any data items subordinate to it, except for items of class index, object, or pointer, are referred to in the following general rules as 'the elements of the operand'.

3) If more than one identifier-1 is specified in a VALIDATE statement, the result of executing this statement is the same as if a separate VALIDATE statement had been written for each identifier-1 in the same order as specified in the statement. If an implicit VALIDATE statement results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, processing resumes at the next implicit VALIDATE statement, if any.

4) The VALIDATE statement is executed in five stages. Each stage is executed for all the elements of the operand before each next stage. Any given stage may be omitted because of the absence of any clause applying to that stage, or because all the elements of the operand failed an earlier stage. The five stages are listed below, together with the clauses that apply to each stage:

   a) Format validation: the DEFAULT clause, the PICTURE clause, the SIGN clause, and the USAGE clause.

   b) Input distribution: the DESTINATION clause.

   c) Content validation: the CLASS clause and the VALUE clause.

   d) Relation validation: the INVALID clause.

   e) Error indication: the VALIDATE-STATUS clause.

   At each stage, the execution of the VALIDATE statement will be affected also by any OCCURS clause, REDEFINES clause, or PRESENT WHEN clause that is specified in the data description entry for any of the elements of the operand.

5) An internal indicator is assigned to each of the elements of the operand whose data description contains (or is associated with, in the case of 88-level entries) a clause that causes the data item to take part in either format

validation, content validation, or relation validation. If any invalid status is detected in a data item as a result of any of these data description clauses, the execution of the VALIDATE statement does not terminate and the content of the invalid data item does not change. Instead, the invalid condition is recorded by setting the data item's internal indicator to one of three distinct values indicating invalid data — designated invalid on format, invalid on content, and invalid on relation.

An elementary item that is invalid on format does not undergo a content check and a data item that is invalid on content does not undergo a relation check.

NOTE Therefore, internal indicators do not change from invalid on format to invalid on content or relation, or from invalid on content to invalid on relation. An indicator is set to indicate an error only if it is in its initial valid state.

If any data description contains an OCCURS clause, internal indicators are assigned independently to each occurrence of the data item, or up to the maximum number of occurrences if the TO phrase is present.

The internal indicators for all of the elements of the operand are assigned the initial valid value at the start of the execution of the VALIDATE statement. For any data items that are subsequently not processed by the VALIDATE statement, the internal indicators are set to a unique value signifying not processed.

6) The validation process consists of five stages that are listed in the following paragraphs. If, during the execution of these stages a fatal exception condition other than EC-DATA-INCOMPATIBLE is set to exist, the execution of the VALIDATE statement ceases and control proceeds as defined for fatal exception conditions. The conditions under which the EC-DATA-INCOMPATIBLE exception condition is set to exist during execution of a VALIDATE statement are specified in 14.6.12.2, Incompatible data. If a nonfatal exception condition is raised, it is processed as defined for nonfatal exception conditions, and, upon completion of exception processing, if any, execution continues as if the exception condition had not been set to exist. The stages of validation are as follows:

   a) Stage one (format validation)

      Each elementary data item is checked for compatibility with the ANY LENGTH, PICTURE, SIGN, and USAGE clauses, as applicable, in the item's data description.

      The integrity of each dynamic-capacity table is verified against the implementor's internal requirements.

      If any data item fails these checks, the data item's internal indicator is set to invalid on format.

      A default value is established for each data item to which the circumstances defined under 13.18.17, DEFAULT clause apply, even if there is no DEFAULT clause associated with the data item.

   b) Stage two (input distribution)

      DESTINATION clauses specified in the data description entries of any of the elements of the operand cause the values of the data items to be moved to the receiving data items according to the rules for the MOVE statement.

   c) Stage three (content validation)

      Each of the operand's data items whose description contains a CLASS clause or is followed by one or more 88-level entries with the VALID or INVALID phrase is now checked for compatibility with the corresponding CLASS definition in SPECIAL-NAMES or the 88-level entries, as described under the VALUE clause.

      If any data item fails any of these checks, its internal indicator is set to invalid on content.

      If both a group item and a subordinate data item are subject to content validation, content validation on the subordinate data item is applied before content validation on the group item.

   d) Stage Four (relation validation)

Each of the operand's data items whose description contains an INVALID clause is now checked according to the rules for relation validation specified under this clause. If any data item fails a relation validation check, its internal indicator is set to invalid on relation.

Any number of INVALID clauses may be present in the same data description entry. If several conditions have been specified, the effect is as though a single INVALID clause had been specified using a condition made by writing each of the original conditions in parentheses joined by the logical connector OR.

If both a group item and a subordinate data item within that group are subject to relation validation, relation validation on the subordinate data item is applied before relation validation on the group item.

e) Stage five (error indication)

If the internal indicator of any of the operand's data items is set to other than its initial valid value, an EC-VALIDATE exception condition is set to exist. If any of these internal indicators is set to invalid on format, an EC-VALIDATE-FORMAT exception condition is set to exist, if any of these internal indicators is set to invalid on content, an EC-VALIDATE-CONTENT exception condition is set to exist, and if any of these internal indicators is set to invalid on relation, an EC-VALIDATE-RELATION exception condition is set to exist.

Any elementary data item whose corresponding internal indicator is set to indicate invalid data is considered to be invalid on format, content, or relation, depending on the value of the internal indicator. A group data item is considered invalid on format if it has a subordinate item that is invalid on format. A group data item is considered invalid on content or relation if it has a subordinate item that is invalid for that reason, or if the group item's own internal indicator shows that it was rejected at that stage.

The data items whose data description entries contain a VALIDATE-STATUS clause with a FOR phrase that references an element of the operand are updated, in accordance with the rules specified in 13.18.61, VALIDATE-STATUS clause, in the order in which their data description entries are specified in the data division.

7) The effect of the OCCURS clause on the execution of the VALIDATE statement is as follows:

a) If the data description entry of identifier-1 contains or is subordinate to an entry that contains an OCCURS clause, identifier-1 itself shall be subscripted and the value of the subscript or subscripts shall specify the particular occurrence of the data item to be processed by the VALIDATE statement.

b) If the data item referenced by identifier-1 is a group data item and there are one or more OCCURS clauses subordinate to it, execution of the VALIDATE statement causes each occurrence of the data item to be processed independently of the other occurrences at all five stages of processing. If there is a VARYING clause associated with an OCCURS clause, any number of counters, defined as data-names in the VARYING clause, are initially set and implicitly incremented for each occurrence, whenever the repeating data item is processed.

c) If any OCCURS clause subordinate to identifier-1 has a DEPENDING phrase, the value of the data item referenced by the DEPENDING phrase is evaluated as soon as the clause is encountered during the format validation stage of the VALIDATE statement. The value obtained establishes, for this and all subsequent stages of processing of the current VALIDATE statement, the number of occurrences of the data item to be processed.

8) If identifier-1 references, in whole or in part, a record in the file section whose FD entry contains a RECORD clause with the VARYING phrase, identifier-1 shall be processed by the VALIDATE statement as though filled or extended on the right, as applicable, by spaces.

9) The data item referenced by identifier-1 shall not contain or overlap any data item specified in a DESTINATION clause contained in or subordinate to the description of identifier-1.

10) The data item referenced by identifier-1 shall not contain or overlap any data item specified in a DEFAULT clause contained in or subordinate to the description of identifier-1 unless that DEFAULT clause follows, within the subordinate items of identifier-1, the description of the data item that the DEFAULT clause references.

14.9.47 WRITE statement

The WRITE statement releases a logical record for an output or input-output file. It also is used for vertical positioning of lines within a logical page.

14.9.47.1 General format

Format 1 (sequential):

```
WRITE { record-name-1 }                [ FROM { identifier-1 } ]
      { FILE file-name-1 }                    { literal-1    }

[ { BEFORE } ADVANCING { { identifier-2 } [ LINE  ] } ]
  { AFTER  }           { { integer-1    } [ LINES ] }
                       { { mnemonic-name-1 }       }
                       { { PAGE            }       }

[ retry-phrase ]

[ WITH LOCK    ]
[ WITH NO LOCK ]

[ AT { END-OF-PAGE } imperative-statement-1                    ]
[    { EOP        }                                            ]
[ NOT AT { END-OF-PAGE } imperative-statement-2                 ]
[        { EOP        }                                        ]

[ END-WRITE ]
```

Format 2 (random):

```
WRITE { record-name-1 }                [ FROM { identifier-1 } ]
      { FILE file-name-1 }                    { literal-1    }

[ retry-phrase ]

[ WITH LOCK    ]
[ WITH NO LOCK ]

[ INVALID KEY imperative-statement-1     ]
[ NOT INVALID KEY imperative-statement-2 ]

[ END-WRITE ]
```

where retry-phrase is described in 14.7.8, RETRY phrase

14.9.47.2 Syntax rules

1) The write file is the file referenced by file-name-1 or by the file-name associated with record-name-1.

2) If the organization of the write file is sequential, format 1 shall be specified.

3) If the organization of the write file is indexed or relative, format 2 shall be specified.

4) If identifier-1 is a function-identifier, it shall reference an alphanumeric, or national function.

5) Record-name-1 is the name of a logical record in the file section of the data division and may be qualified.

6) If record-name-1 is specified, identifier-1 or literal-1 shall be valid as a sending operand in a MOVE statement specifying record-name-1 as the receiving operand.

7) If the FILE phrase is specified, the FROM phrase shall also be specified and:

   a) identifier-1 shall be valid as a sending operand in a MOVE statement;

   b) literal-1 shall be an alphanumeric, boolean, or national literal and shall not be a figurative constant.

8) If the FILE phrase is specified, the description of identifier-1, including its subordinate data items, shall not contain a data item described with a USAGE OBJECT REFERENCE clause.

9) If identifier-1 references a bit data item other than an intrinsic function and the FILE phrase is specified, identifier-1 shall be described such that:

   a) subscripting and reference modification in identifier-1 consist of only fixed-point numeric literals or arithmetic expressions in which all operands are fixed-point numeric literals and the exponentiation operator is not specified; and

   b) it is aligned on a byte boundary.

10) If identifier-1 references an intrinsic function and the FILE phrase is specified, identifier-1 shall reference an alphanumeric or national function.

11) If identifier-1 references an intrinsic function and the FILE phrase is not specified, identifier-1 shall reference an alphanumeric, boolean, or national function.

12) The file description entry associated with the write file shall not contain the REPORT clause and shall not be a sort-merge file description entry.

13) If the file description entry associated with the write file contains the LINAGE clause, mnemonic-name-1 shall not be specified.

14) Identifier-2 shall reference an integer data item.

15) Integer-1 shall be positive or zero.

16) When mnemonic-name-1 is specified, the name is associated with a feature-name specified by the implementor. Mnemonic-name-1 is defined in the SPECIAL-NAMES paragraph of the environment division.

17) The phrases ADVANCING PAGE and END-OF-PAGE shall not both be specified in a single WRITE statement.

18) If the END-OF-PAGE or the NOT END-OF-PAGE phrase is specified, the LINAGE clause shall be specified in the file description entry associated with the write file.

19) The words END-OF-PAGE and EOP are equivalent.

20) If record-name-1 is defined in a containing program and is referenced in a contained program, the file description entry for the file-name associated with record-name-1 shall contain a GLOBAL clause.

21) If automatic locking has been specified for the write file, neither the WITH LOCK phrase nor the WITH NO LOCK phrase shall be specified.

14.9.47.3 General rules

ALL FILES

1) The write file connector is the file connector associated with the write file. If the access mode of the write file is sequential, the open mode of the write file connector shall be extend or output. Otherwise, the open mode of the write file connector shall be I-O or output. If the open mode is not as described, the execution of the WRITE statement is unsuccessful and the I-O status for the write file connector is set to '48'.

2) The logical record released by the successful execution of the WRITE statement is no longer available in the record area unless the file-name associated with record-name-1 is specified in a SAME RECORD AREA clause. The logical record is also available as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as the file associated with record-name-1.

3) The result of the execution of a WRITE statement specifying record-name-1 and the FROM phrase is equivalent to the execution of the following statements in the order specified:

   a) The statement:

      ```
      MOVE identifier-1 TO record-name-1
      ```

      or

      ```
      MOVE literal-1 TO record-name-1
      ```

      according to the rules specified for the MOVE statement.

   b) The same WRITE statement without the FROM phrase.

   NOTE 14.6.9, Overlapping operands, and 14.9.24, MOVE statementt, general rules, apply to any cases in which the storage area identified by identifier-1 and the record area associated with record-name-1 share any part of their storage areas. The result of execution of the WRITE statement is undefined if the result of execution of the implicit MOVE statement described in general rule 3a is undefined.

4) The figurative constant SPACE when specified in the WRITE statement references one alphanumeric space character.

5) For a WRITE statement with the FILE phrase in which identifier-1 references a record description entry that is associated with file-name-1, then:

   a) When identifier-1 references a record description entry subordinate to the file description entry identified by file-name-1, the result of execution of the WRITE statement is equivalent to the result of execution of WRITE identifier-1.

   b) When identifier-1 references a record description entry that is not subordinate to the file description entry referenced by file-name-1, the result of execution of the WRITE statement is equivalent to the result that would be obtained if the record description entry were subordinate to the file description entry referenced by file-name-1 and the statement WRITE identifier-1 had been executed.

6) For a WRITE statement with the FILE phrase in which identifier-1 does not reference a record description entry that is associated with file-name-1, the result of execution is equivalent to the execution of the following in the order specified:

a) The statement:

```
MOVE identifier-1 TO implicit-record-1
```

or

```
MOVE literal-1 TO implicit-record-1
```

b) The statement:

```
WRITE implicit-record-1
```

where implicit-record-1 refers to the record area for file-name-1 and is treated:

a) when identifier-1 references an intrinsic function, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same class, category, usage, and length as the returned value of the intrinsic function, or

b) when identifier-1 does not reference an intrinsic function, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same description as identifier-1, or

c) when literal-1 is specified, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same class, category, usage, and length as literal-1.

NOTE 14.6.9, Overlapping operands, and 14.9.24, MOVE statementt, general rules, apply to any cases in which the storage area identified by identifier-1 and the record area associated with implicit-record-1 share any part of their storage areas. The result of execution of the WRITE statement is undefined if the result of execution of the implicit MOVE statement described in general rule 6 is undefined.

7) After the successful execution of a WRITE statement, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name-1 is not available except as specified by the SAME RECORD AREA clause.

8) If the locking mode of the write file connector is single record locking, any record lock associated with that file connector is released by the execution of the WRITE statement.

9) If record locks have an effect for the write file connector and the WITH LOCK phrase is specified, the record lock associated with the record written is set when the execution of the WRITE statement is successful.

10) The file position indicator is not affected by the execution of a WRITE statement.

11) The execution of a WRITE statement causes the value of the I-O status of the write file connector to be updated to a value given in 9.1.12, I-O status.

12) The successful execution of a WRITE statement releases a logical record to the operating environment.

NOTE Logical records in relative and sequential files may have a length of zero. Logical records in an indexed file shall always be long enough to contain the record keys.

13) When record-name-1 is specified, if the number of bytes to be written to the file is greater than the number of bytes in record-name-1, the content of the bytes that extend beyond the end of record-name-1 are undefined.

14) If the execution of a WRITE statement is unsuccessful, the write operation does not take place, the content of the record area is unaffected, and the I-O status of the write file connector is set to a value indicating the cause of the condition as specified in the following general rules. The transfer of control depends on other clauses and the value of the I-O status as described in 9.1.12, I-O status and 9.1.13, Invalid key condition.

15) The number of bytes in the runtime representation of literal-1, the data item referenced by identifier-1, or the record referenced by record-name-1 after any changes made to the record length by the FORMAT clause shall not be larger than the largest or smaller than the smallest number of bytes allowed by the RECORD IS VARYING clause associated with file-name-1 or the file-name associated with record-name-1. If this rule is violated, the execution of the WRITE statement is unsuccessful and the I-O status of the write file connector is set to '44'.

16) The RETRY phrase is used to control the behavior of the WRITE statement for files opened for file sharing for the case where resources needed to write a record are locked by another run unit. The I-O status is set in accordance with the rules in 14.7.8, RETRY phrase.

SEQUENTIAL FILES

17) The successor relationship of a sequential file is established by the order of execution of WRITE statements when the physical file is created. The relationship does not change except when records are added to the end of a physical file.

18) When the organization of the write file connector is sequential and the open mode is extend, the execution of the WRITE statement will add records to the end of the physical file as though the open mode of the file connector were output. If there are records in the physical file, the first record written after the execution of the OPEN statement with the EXTEND phrase is the successor of the last record in the physical file.

19) If two or more file connectors for a sequential file add records by sharing the physical file after opening it in extend mode, the added records follow the records present in the physical file when it was opened, but are otherwise in an undefined order.

20) When an attempt is made to write beyond the externally-defined boundaries of the physical file, the execution of the WRITE statement is unsuccessful and the I-O status value of the write file connector is set to '34'.

21) If the end of reel/unit is recognized and the externally defined boundaries of the physical file have not been exceeded, a reel/unit swap occurs and the current volume pointer is updated to point to the next reel/unit existing for the physical file.

22) Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing shall be provided by the implementor to act as if the user has specified AFTER ADVANCING 1 LINE. If the physical file does not support vertical positioning, the ADVANCING and END-OF-PAGE phrases are ignored. If the physical file does support vertical positioning and the ADVANCING phrase is specified, advancing is provided as follows:

    a) If integer-1 or the value of the data item referenced by identifier-2 is positive, the representation of the printed page is advanced the number of lines equal to that value.

    b) If the value of the data item referenced by identifier-2 is negative, the results are undefined.

    c) If integer-1 or the value of the data item referenced by identifier-2 is zero, no repositioning of the representation of the printed page is performed.

    d) If mnemonic-name-1 is specified, the representation of the printed page is advanced according to the rules specified by the implementor for that hardware device.

    e) If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to general rule 22a, 22b, 22c, and 22d above.

    f) If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to general rule 22a, 22b, 22c, and 22d above.

g) If PAGE is specified and the LINAGE clause is specified in the associated file description entry, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page. The repositioning is to the first line that may be written on the next logical page as specified in the LINAGE clause.

h) If PAGE is specified and the LINAGE clause is not specified in the associated file description entry, the record is presented on the physical page before or after (depending on the phrase used) the device is repositioned to the next physical page. If physical page has no meaning in conjunction with a specific device, advancing will be provided as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

23) If the LINAGE clause is specified in the file description entry of the associated file, an end-of-page condition occurs when the lines written by a WRITE statement do not fit within the current page body. This occurs when:

a) The logical end of the representation of the printed page is reached. This occurs when the associated LINAGE-COUNTER is equal to or exceeds the page size. If the AFTER phrase is specified or implied, the device is repositioned to the first line that may be written on the next logical page and the logical record is presented on that line. If the BEFORE phrase is specified, the logical record is presented and the device is repositioned to the first line that may be written on the next logical page.

b) The FOOTING phrase is specified in the LINAGE clause and the execution of the WRITE statement causes printing or spacing within the footing area of a page body. This occurs when the associated LINAGE-COUNTER is equal to or exceeds the current value of the footing start and is less than the page size.

24) When an end-of-page condition occurs, the following actions take place:

a) If the end-of-page condition was caused by the action in general rule 23a, the EC-I-O-EOP-OVERFLOW exception condition is set to exist. If the end-of-page condition was caused by the action in general rule 23b, the EC-I-O-EOP exception condition is set to exist.

b) If the END-OF-PAGE phrase is specified, control is transferred to imperative-statement-1 and execution continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the WRITE statement. The NOT END-OF-PAGE phrase, if specified, is ignored.

25) If, during the successful execution of a WRITE statement with the NOT END-OF-PAGE phrase, the end-of-page condition does not occur, control is transferred to imperative-statement-2 after execution of the input-output operation and execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of execution of imperative-statement-1, control is transferred to the end of the WRITE statement. The END-OF-PAGE phrase, if specified, is ignored.

RELATIVE FILES

26) The WRITE statement proceeds as follows:

a) If the access mode of the write file connector is sequential, the successful execution of the WRITE statement causes a record to be released to the operating environment. If the open mode of the write file connector is output, the first record released after the OPEN is 1. If the open mode is extend, the first record released after the OPEN is assigned a record number that is one greater than the highest relative record number existing in the physical file. Subsequent records released have relative record numbers that are ascending ordinal numbers. If the physical file is shared and the open mode is extend, the record numbers are not necessarily consecutive. Otherwise, they are consecutive. If the RELATIVE KEY clause is specified for file-name-1 or the file-name associated with record-name-1, the relative record number of the record being released is moved into the relative key data item by the operating environment during

execution of the WRITE statement according to the rules for the MOVE statement. If the maximum numeric value allowed for the relative key data item is exceeded by the relative record number that would be generated by a successful WRITE operation, the WRITE statement is unsuccessful, the invalid key condition exists, and the I-O status for the write file connector is set to '24'.

b) If the access mode of the write file connector is random or dynamic, prior to the execution of the WRITE statement the value of the relative key data item shall be initialized by the runtime element with the relative record number to be associated with the record that is to be written. If there is no record in the physical file with a relative record number that matches the relative key value, the record is released to the operating environment. If a record in the file matches, the execution of the WRITE statement is unsuccessful, the invalid key condition exists, and the I-O status for the write file connector is set to '22'.

27) When a relative file is opened with the file connector in the extend mode, records are inserted into the file through that file connector. The first record released to the operating environment has a relative record number one greater than the highest relative record number existing in the file. Subsequent records released to the operating environment have consecutively higher relative record numbers. If the RELATIVE KEY clause is specified for file-name-1 or the file-name associated with record-name-1, the relative record number of the record being released is moved into the relative key data item by the operating environment during execution of the WRITE statement according to the rules for the MOVE statement.

28) If two or more file connectors for a relative file add records by sharing the file after opening it in extend mode, the relative key values returned are ascending, but not necessarily consecutive.

29) When a relative file is opened in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. Prior to the execution of the WRITE statement, the value of the relative key data item shall be initialized by the runtime element with the relative record number to be associated with the record that is to be written. That record is then released to the operating environment by the execution of the WRITE statement.

30) The invalid key condition exists under the following circumstances, regardless of any locks that are associated with the record being accessed:

a) When the value of the relative key data item specifies a record that already exists in the file, the I-O status associated with the write file connector is set to '22'.

b) When an attempt is made to write beyond the externally defined boundaries of the file, the I-O status associated with the write file connector is set to '24'.

c) When the number of significant digits in the relative record number is larger than the size of the relative key data item specified for the file, the I-O status associated with the write file connector is set to '24'.

INDEXED FILES

31) Successful execution of a WRITE statement causes the content of the record area to be released. The operating environment utilizes the contents of the record keys in such a way that subsequent access of the record may be made based upon any of these specified record keys.

32) The comparison used for ensuring uniqueness or for ordering records in the physical file is based on the collating sequence for the file according to the rules for a relation condition.

33) The value of the prime record key shall not be equal to the value of the prime record key of any record existing in the file.

34) The data item specified as the prime record key shall be set by the runtime element to the desired value prior to the execution of the WRITE statement.

35) If the access mode of the write file connector is sequential, records shall be released to the operating environment through that file connector in ascending order of prime record key values according to the

collating sequence of the file. When the open mode is extend, the first record released to the operating environment shall have a prime record key whose value is greater than the highest prime record key value existing in the physical file when it was opened through that file connector and each subsequent record released to the operating environment through that file connector shall have a prime record key whose value is greater than the highest prime record key value written referencing this file connector. If the record is not in the sequence above, the execution of the WRITE statement is unsuccessful, the invalid key condition exists, and the I-O status for the write file connector is set to '21'.

36) If the access mode of the write file connector is random or dynamic, WRITE statements may release records to the operating environment through that connector in any order.

37) When the ALTERNATE RECORD KEY clause is specified in the file control entry associated with the write file connector, the value of the alternate record key may be nonunique only if the DUPLICATES phrase is specified for that data item. In this case the operating environment provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which the operating environment actually writes the record into the physical file. If the DUPLICATES phrase is not specified and the alternate key value is nonunique, the execution of the WRITE statement is unsuccessful, the invalid key condition exists, and the I-O status of the write file connector is set to '22'.

38) The invalid key condition exists under the following circumstances. The comparison for equality for record keys is based on the collating sequence for the file according to the rules for a relation condition. Any record locks associated with the record being accessed are ignored in detection of these exceptions.

    a) When the write file connector is open for output or extend in the sequential access mode and the value of the prime record key is not greater than the value of the prime record key of the last record written through that file connector, the I-O status associated with the write file connector is set to '21'.

    b) When the value of the prime record key of the record to be written is equal to the value of the prime record key of any record existing in the file, the I-O status associated with the write file connector is set to '22'.

    c) When an alternate record key of the record to be written does not allow duplicates and the value of that alternate record key is equal to the value of the corresponding alternate record key of a record in the file, the I-O status associated with the write file connector is set to '22'.

    d) When the record that is to be released to the operating environment would reside beyond the externally defined boundaries of the physical file, the I-O status associated with the write file connector is set to '24'.

15 Intrinsic functions

15.1 General

Each intrinsic function definition specifies:

1) the name and description of the function
2) the type of the function
3) the general format of the function
4) the arguments, if any
5) the returned value.

See 8.4.2.2, Function-identifier, for rules and explanations on the referencing of functions.

15.2 Types of functions

Types of intrinsic functions are:

1) Alphanumeric functions. These are of the class and category alphanumeric. The number of character positions in this data item is specified in the function definition. Alphanumeric functions have an implicit usage display. Unless stated otherwise in the definition of a function, the data item is represented in the alphanumeric coded character set in effect when the function is referenced at runtime.

2) Boolean functions. These are of the class and category boolean. The number of boolean positions in this data item is specified in the function definition. Boolean functions have an implicit usage bit.

3) National functions. These are of the class and category national. The number of character positions in this data item is specified in the function definition. National functions have an implicit usage national. Unless stated otherwise in the definition of a function, the data item is represented in the national coded character set in effect when the function is referenced at runtime.

4) Numeric functions. These are of the class and category numeric. A numeric function has an operational sign.

5) Integer functions. These are of the class and category numeric. An integer function has an operational sign and no digits to the right of the decimal point.

6) Index functions. These are of the class and category index.

15.3 Arguments

Arguments specify values used in the evaluation of a function. Arguments are specified in the function-identifier. The definition of a function specifies the number of arguments required, which may be zero, one, or more. For some functions, the number of arguments may be variable. The order in which arguments are specified in a function-identifier determines the interpretation given to each value in arriving at the function value.

Arguments may be required to have a certain class or a subset of a certain class, to be a keyword, a type declaration, or a mnemonic-name. The types of argument are:

1) Alphabetic. An elementary data item of the class alphabetic or an alphanumeric literal containing only alphabetic characters shall be specified. The size associated with the argument may be used in determining the value of the function.

2) Alphanumeric. A data item of the class alphabetic or alphanumeric or an alphanumeric literal shall be specified. The size associated with the argument may be used in determining the value of the function. Strongly-typed group items are treated as though they were of class and category alphanumeric, unless they are prohibited as arguments of a function.

3) Boolean. A bit group item, a boolean expression or literal, or an elementary boolean data item shall be specified. The size associated with the argument may be used in determining the value of the function.

4) Index. An index data item shall be specified. The size associated with the argument may be used in determining the value of the function.

5) Integer. An arithmetic expression that will always result in an integer value or an integer data item shall be specified. The value of the arithmetic expression, including operational sign, is used in determining the value of the function.

6) Keyword. A keyword shall be specified in accordance with the function definition.

7) Locale-name. A locale-name defined in the SPECIAL-NAMES paragraph shall be specified. The locale associated with the locale-name may be used in determining the value of the function.

8) National. A national group item, a national literal, or an elementary data item of class national shall be specified. The size associated with the argument may be used in determining the value of the function.

9) Numeric. An arithmetic expression or a numeric data item shall be specified. The value of the arithmetic expression is used in determining the value of the function.

10) Object. An object reference shall be specified; the predefined object reference SUPER shall not be specified. The size associated with the argument may be used in determining the value of the function.

11) Ordering-name. An ordering-name defined in the SPECIAL-NAMES paragraph shall be specified. The ordering table associated with the ordering-name may be used in determining the value of the function.

12) Pointer. A data item of class pointer shall be specified. The size associated with the argument may be used in determining the value of the function.

13) Type declaration. A type-name shall be specified. The size associated with the type declaration may be used in determining the value of the function.

A variable-length group shall be referenced as an argument to a function only when explicitly permitted in the function definition.

If any function permits a type declaration as an argument, the type declaration shall not describe a variable-length group.

The rules for a function may place constraints on the permissible values for arguments in order to permit meaningful determination of the function's value. If the evaluation of an argument results in an incorrect value for that argument according to the rules specified in the function definition and no exception condition was raised during item identification or expression evaluation, the EC-ARGUMENT-FUNCTION exception condition is set to exist. If an exception condition is raised during item identification or expression evaluation, that exception condition is raised, not EC-ARGUMENT-FUNCTION. If the EC-ARGUMENT-FUNCTION exception condition is set to exist and checking for EC-ARGUMENT-FUNCTION is not enabled, the implementor defines the result of the function reference.

NOTE An example of another exception condition that might be raised is EC-SIZE-OVERFLOW during evaluation of a subscript or arithmetic expression.

When the definition of a function permits an argument to be repeated a variable number of times, a table may be referenced by specifying the data-name and any qualifiers that identify the table, followed immediately by subscripting where one or more of the subscripts is the word ALL.

If the subscript ALL is specified for a dynamic-capacity table, the range of values of the subscript is from 1 to the current capacity of the table.

When ALL is specified as a subscript, the effect is as if each table element associated with that subscript position were specified. The order of the implicit specification of each occurrence is from left to right, with the first (or leftmost) specification being the identifier with each subscript specified by the word ALL replaced by one, the next specification being the same identifier with the rightmost subscript specified by the word ALL incremented by one. This process continues with the rightmost ALL subscript being incremented by one for each implicit specification until the rightmost ALL subscript has been incremented through its range of values. If there are any additional ALL subscripts, the ALL subscript immediately to the left of the rightmost ALL subscript is incremented by one, the rightmost ALL subscript is reset to one and the process of varying the rightmost ALL subscript is repeated. The ALL subscript to the left of the rightmost ALL subscript is incremented by one through its range of values. For each additional ALL subscript, this process is repeated in turn until the leftmost ALL subscript has been incremented by one through its range of values. If the ALL subscript is associated with a data item described with an OCCURS DEPENDING ON clause, the range of values is determined by the object of the OCCURS DEPENDING ON clause. The evaluation of an ALL subscript shall result in at least one argument, otherwise the result of the reference to the function-identifier is undefined.

15.3.1 Format arguments to date and time functions

The date, time, and combined date and time formats described below are representations of the corresponding formats in ISO 8601. Certain intrinsic functions accept literals representing these formats as arguments.

A single format describes a variety of information. For example, a basic calendar date format includes a representation of the year, a representation of the month of that year, and a representation of the day of that month. The portions of the format are considered subfields of the format. The portions of the values that correspond to the subfields of the format are considered subfields of the values.

Table 22, Table of date and time formats, in section 15.3.1.4, Examples of date and time formats, summarizes the formats and provides examples of values associated with them.

15.3.1.1 Date formats

COBOL supports six date formats: basic and extended formats for calendar dates, basic and extended formats for ordinal dates, and basic and extended formats for week dates.

Examples throughout this section use a date of Wednesday, February 15, 1995, for illustration of the results of application of the formats.

15.3.1.1.1 Calendar date formats

The basic calendar date format contains eight characters: four uppercase 'Y' characters representing the year subfield; two uppercase 'M' characters representing the month subfield; and two uppercase 'D' characters representing the day subfield.

The extended calendar date format contains ten characters: four uppercase 'Y' characters representing the year subfield; a hyphen; two uppercase 'M' characters representing the month subfield; a hyphen; and two uppercase 'D' characters representing the day-of-month subfield. The two hyphens appear in the data associated with the extended calendar date format.

15.3.1.1.2 Permissible values for data associated with calendar date formats

The year subfield of the data (corresponding to 'YYYY' in the format) shall contain a value greater than 1600 and less than or equal to 9999.

The month subfield of the data (corresponding to 'MM' in the format) shall contain a value from 01 through 12 inclusive.

The day-of-month subfield of the data (corresponding to 'DD' in the format) shall contain a value from 01 through 28, 29, 30 or 31, depending on the contents of the month subfield and the year subfield of the data.

15.3.1.1.3 Ordinal date formats

The basic ordinal date format contains seven characters: four uppercase 'Y' characters representing the year subfield and three uppercase 'D' characters representing the day-of-year subfield.

The extended ordinal date format contains eight characters: four uppercase 'Y' characters representing the year subfield; a hyphen; and three uppercase 'D' characters representing the day-of-year subfield. The hyphen appears in the data associated with the extended ordinal date format.

15.3.1.1.4 Permissible values for data associated with ordinal date formats

The year subfield of the data (corresponding to 'YYYY' in the format) shall contain a value greater than 1600 and less than or equal to 9999.

The day-of-year subfield of the data (corresponding to 'DDD' in the format) shall contain a value from 1 through either 365 when the contents of the year subfield represent a common year or 366 when the contents of the year subfield represent a leap year.

15.3.1.1.5 Week date formats

The basic week date format contains eight characters: four uppercase 'Y' characters representing the year subfield; an uppercase 'W' character; two lowercase 'w' characters representing the week-of-year subfield; and a single uppercase 'D' character representing the day-of-week subfield. The uppercase 'W' character appears in the corresponding position of data associated with a basic week-date format.

The extended week date format contains ten characters: four uppercase 'Y' characters representing the year subfield; a hyphen; an uppercase 'W' character; two lowercase 'w' characters representing the week-of-year subfield; a hyphen; and a single uppercase 'D' character representing the day-of-week subfield. The two hyphens and the uppercase 'W' character appear in the corresponding positions of data associated with an extended week-date format.

15.3.1.1.6 Permissible values for data associated with week date formats

The year subfield of the data (corresponding to 'YYYY' in the format) shall contain a value greater than 1600 and less than or equal to 9999.

The first week of a given year is the week that includes January 4 of that year. The last week of a given year is the week immediately preceding the first week of the following year.

The week-of-year subfield of the data (corresponding to ww in the format) shall contain a value from 01 through 53 inclusive if the year subfield represents a common year and January 4 of that year falls on Sunday, or if the year subfield represents a leap year and January 4 of that year falls on a Saturday or Sunday. Otherwise, the week-of-year subfield shall contain a value from 01 through 52 inclusive.

> NOTE It is possible for dates at the beginning of January to be represented as occurring in the last week of the previous year, for example, if January 4 occurs on a Monday.

The day-of-week subfield of the data (corresponding to 'D' in the format) shall contain a value from 1 through 7 inclusive, representing Monday through Sunday respectively.

15.3.1.2 Time formats

COBOL supports four formats for local time of day ('local time'), four formats for UTC (Coordinated Universal Time) ('UTC time'), and four formats for local time with offset from UTC ('offset time').

A portion of the specifications for these formats is common to all of them; the specifications that are shared by all of them are referred to as 'common time format'.

15.3.1.2.1 Common time formats

15.3.1.2.1.1 Common time formats with integer seconds representation

The basic common time format with integer seconds representation contains six characters: two lowercase 'h' characters representing the hours subfield; two lowercase 'm' characters representing the minutes subfield; and two lowercase 's' characters representing the seconds subfield.

The extended common time format with integer seconds representation contains eight characters: two lowercase 'h' characters representing the hours subfield; a colon character; two lowercase 'm' characters representing the minutes subfield; a colon character; and two lowercase 's' characters representing the seconds subfield. The two colon characters appear in the data associated with an extended common time format.

15.3.1.2.1.2 Common time formats with fractional seconds representation

The basic common time format with fractional seconds contains a minimum of eight characters: two lowercase 'h' characters representing the hours subfield; two lowercase 'm' characters representing the minutes subfield; two lowercase 's' characters representing the integer portion of the seconds subfield; a decimal separator; and at least one lowercase 's' character representing a digit in the decimal fraction portion of the seconds subfield. The decimal separator does not appear in the data associated with a basic common time format with fractional seconds representation.

The extended common time format with fractional seconds contains a minimum of ten characters: two lowercase 'h' characters representing the hours subfield; a colon character; two lowercase 'm' characters representing the minutes subfield; a colon character; two lowercase 's' characters representing the integer portion of the seconds subfield; a decimal separator; and at least one lowercase 's' character representing a digit in the decimal fraction portion of the seconds subfield. The two colon characters and the decimal separator appear in the data associated with an extended common time format with fractional seconds representation.

NOTE Representations of hours and minutes with digits to the right of the decimal point are not supported in COBOL time formats.

The implementor defines the maximum number of digit positions that may be specified in the decimal fraction portion of the seconds subfield of a time format; that maximum shall be greater than or equal to nine.

A comma is used as a decimal separator in a time format if the DECIMAL-POINT IS COMMA clause is specified.

A period is used as a decimal separator in a time format if the DECIMAL-POINT IS COMMA clause is not specified.

NOTE ISO 8601 specifies that the preferred decimal separator is the comma, which is contrary to normal COBOL practice; the latter is followed here.

15.3.1.2.1.3 Permissible values for data associated with common time formats

The hours subfield of the data (corresponding to hh in a common time format) shall contain a value from 00 to 23 inclusive.

NOTE ISO 8601 allows the choice between whether midnight is at the beginning of the day (00:00:00) or at the end of the day (24:00:00). The COBOL intrinsic functions that accept formatted time values in arguments do not accept 24 in the hours subfield.

The minutes subfield of the data (corresponding to mm in a common time format) shall contain a value from 00 to 59 inclusive.

The seconds subfield of the data shall contain a value that is greater than or equal to 00 and is less than 60 when the LEAP-SECOND directive with the OFF phrase is in effect.

The seconds subfield of the data shall contain a value that is greater than or equal to 00 and less than 61 when the LEAP-SECOND directive with the ON phrase is in effect.

15.3.1.2.2 Local time formats

Basic and extended local time formats contain only that information which is common to all time formats, as described in 15.3.1.2.1, Common time formats.

15.3.1.2.3 UTC time formats

Basic and extended UTC time formats consist of basic and extended common time formats followed by a single uppercase 'Z' character.

15.3.1.2.4 Offset time formats

Basic and extended offset time formats consist of basic and extended common time formats followed by an offset subformat. Offset subformats are described separately below.

Special considerations regarding the offset subformat of an extended offset time format are described in 15.3.1.2.4.1, Offset subformats.

15.3.1.2.4.1 Offset subformats

The offset subformat of a basic offset time format contains five characters: a plus sign; two lowercase 'h' characters representing the hours portion of the offset from UTC (the offset-hours subfield) and two lowercase 'm' characters representing the minutes portion of the offset from UTC (the offset-minutes subfield).

The offset subformat of an extended offset time format contains six characters: a plus sign; two lowercase 'h' characters representing the offset-hours subfield; a colon character; and two lowercase 'm' characters representing the offset-minutes subfield.

In the data associated with an offset subformat, the character position that corresponds to the plus sign in the subformat shall contain one of the following values;

1) A plus sign to indicate that the common time portion of the data is adjusted downward by the offset values to represent UTC;

2) A minus sign to indicate that the common time portion of the data is adjusted upward by the offset values to represent UTC; or

3) A zero to indicate that offset from UTC was not available on the system on which the information was recorded.

If the data contains a zero in that character position, the offset-hours and offset-minutes subfields of the data shall both contain zero.

> NOTE 1 The value "1014278124Z" interpreted according to the format 'hhmmss.ssssZ' and the value "0514278124-0500" interpreted according to the format 'hhmmss.ssss+hhmm' represent the same time, namely, 10:14:27.8124 Coordinated Universal Time.
>
> NOTE 2 The returned value rules for the CURRENT DATE and WHEN COMPILED functions describe cases in which a zero in the sign position of an offset subformat can occur.

If the values contained in offset-hours and offset-minutes subfields of data in a receiving field associated with an offset subformat are both zero, the position corresponding to the plus sign in the format shall contain a plus sign. For offset-time values used as sending fields, the position corresponding to the plus sign in the format may contain a zero provided that the offset-hours and offset-minutes subfields of the data both contain zeroes.

> NOTE When local time is UTC, the data corresponding to the offset subformat of a basic offset date format contains +0000, and the data corresponding to the offset subformat of an extended offset date format contains +00:00.

The colon character in the offset subformat of an extended offset time format appears in the data associated with the format.

15.3.1.2.5 Permissible values for data associated with offset time formats

The values that may be contained in the common time format portion of a basic or extended offset time format are described in 15.3.1.2.1.3, Permissible values for data associated with common time formats.

The two colon characters in the common time portion of an extended offset time format, and the colon character in the offset subformat portion of an extended offset time format, appear in the data associated with that format.

The offset-hours subfield of the data associated with a basic or extended offset time format shall contain a value from 00 to 23 inclusive.

The offset-minutes subfield of the data associated with a basic or extended offset time format shall contain a value from 00 to 59 inclusive.

15.3.1.3 Combined date and time formats

A basic combined date and time format consists of a basic date format followed by an uppercase 'T' character followed by a basic time format (for example, YYYYMMDDThhmmss.sss+hhmm).

An extended combined date and time format consists of an extended date format, followed by an uppercase 'T' character, followed by an extended time format (for example, YYYY-MM-DDThh:mm:ss.sssss+hh:mm).

Combinations of basic date formats with extended time formats, or of extended date formats with basic time formats, are not allowed.

The uppercase 'T' character that occurs in both basic and extended combined and time formats, which serves to separate the date format portion from the time format portion, appears in the data associated with the format.

15.3.1.4 Examples of date and time formats

Table 22 summarizes permissible date and time formats and the values that may be associated with them.

All date format examples use Wednesday, February 15, 1995, and all time format examples use 05:14:27.812479168304 Eastern Standard Time (on the same calendar date) for illustrative purposes.

> NOTE Future enhancements to standard intrinsic functions might include arguments that are not specified in the current standard. The possibility of syntactic conflicts introduced by implementor extensions that specify additional arguments to such functions is lessened considerably if the implementor-defined arguments are either keyword arguments or are identified by keywords.

Table 22 — Table of date and time formats

Type of format	Format	Example value	Notes
Basic calendar date	YYYYMMDD	19950215	
Extended calendar date	YYYY-MM-DD	1995-02-15	
Basic ordinal date	YYYYDDD	1995046	
Extended ordinal date	YYYY-DDD	1995-046	
Basic week date	YYYYWwwD	1995W063	
Extended week date	YYYY-Www-D	1995-W06-3	
Basic local time (integer seconds)	hhmmss	051427	

Table 22 — Table of date and time formats (Continued)

Type of format	Format	Example value	Notes
Extended local time (integer seconds)	hh:mm:ss	05:14:27	
Basic local time (fractional seconds)	hhmmss.sssssssss	051427.812479168	1
Extended local time (fractional seconds)	hh:mm:ss.sssssssss	05:14:27.812479168	1
Basic UTC time (integer seconds)	hhmmssZ	101427Z	
Extended UTC time (integer seconds)	hh:mm:ssZ	10:14:27Z	
Basic UTC time (fractional seconds)	hhmmss.ssssssssZ	101427.812479168Z	1
Extended UTC time (fractional seconds)	hh:mm:ss.ssssssssZ	10:14:27.812479168Z	1
Basic offset time (integer seconds)	hhmmss+hhmm	051427-0500	
Extended offset time (integer seconds)	hh:mm:ss+hh:mm	05:14:27-05:00	
Basic offset time (fractional seconds)	hhmmss.sssssssss+hhmm	051427.812479168-0500	1
Extended offset time (fractional seconds)	hh:mm:ss.sssssssss+hh:mm	05:14:27.812479168-05:00	1
Combined basic date and time format (integer seconds)	YYYYMMDDThhmmss	19950215T051427	2
Combined extended date and time format (integer seconds)	YYYY-MM-DDThh:mm:ss+hh:mm	1995-02-15T05:14:27-05:00	2
Combined basic date and time format (fractional seconds)	YYYYMMDDThhmm.ss+hhmm	19950215T051427.81+0500	2,3

NOTE 1 In time formats and combined date and time formats that reflect fractional seconds, at least one 's' character shall appear immediately to the right of the decimal separator, and the implementor shall provide support for at least nine 's' characters to the right of the decimal separator. A comma is used as the decimal separator in both formats and the associated data if the DECIMAL-POINT IS COMMA clause is specified; otherwise, a period is used as the decimal separator. The examples shown use nine 's' characters to the right of the decimal separator; corresponding formats with from one to eight such characters are permitted but not illustrated. The implementor may specify that more than nine 's' characters may appear to the right of the decimal point. o the right of the decimal separator are allowed.

NOTE 2 Basic combined date and time formats consist of any basic date format followed by the character 'T' followed by any basic time format. Extended combined date and time formats consist of any extended date format followed by the character 'T' followed by any extended time format. The full set of permitted combined date and time formats can be easily derived and is not shown here.

15.4 Returned values

The evaluation of a function produces a returned value in a temporary elementary data item. The type of a function identifies the type of the returned value as specified in 15.2, Types of functions.

15.4.1 Numeric and integer functions

The returned value rules for certain integer and numeric intrinsic functions contain one or more equivalent arithmetic expressions. An equivalent arithmetic expression is a formal definition that defines the relationship among a function, its arguments, and its returned value. In the presentation of the equivalent arithmetic expressions where there is a variable number of occurrences of an argument, the rules may contain an equivalent arithmetic expression for one, two, and n occurrences.

The returned value of numeric and integer functions depends on the mode of arithmetic in effect, native, standard, standard-binary, or standard-decimal, and on whether an equivalent arithmetic expression is specified for the function.

When standard arithmetic, standard-binary arithmetic, or standard-decimal arithmetic is in effect, the returned value for numeric and integer functions is contained in a temporary standard data item in the intermediate form defined for the arithmetic mode in effect. With the exception of the DATE-TO-YYYYMMDD function when argument-3 is not specified, the DAY-TO-YYYYDDD function when argument-3 is not specified, the RANDOM function when no argument is specified, and the YEAR-TO-YYYY function when argument-3 is not specified, the returned value is the same for all instances of a given function within a single execution of the runtime element so long as the value and order of the arguments, the collating sequence, and the locale are unchanged.

When native arithmetic is in effect, the characteristics and representation of the returned value are defined by the implementor.

When native arithmetic is in effect and an equivalent arithmetic expression is specified, the value returned is an implementor-defined approximation of the value of that expression.

NOTE The result of an equivalent arithmetic expression is implementor-defined if any one or more of the following applies:

1) Native arithmetic is in effect.

2) One or more of the arguments are implementor-defined.

3) One or more of the arithmetic expressions that compose the equivalent arithmetic expression produces implementor-defined results.

4) The result is explicitly implementor-defined.

When standard arithmetic, standard-binary arithmetic, or standard-decimal arithmetic is in effect and an equivalent arithmetic expression is specified:

1) the returned value shall equal the value of the equivalent arithmetic expression.

   NOTE As a result, the relation condition

   function-identifier = equivalent-arithmetic-expression

   will evaluate to true.

2) the result of an equivalent arithmetic expression is implementor-defined if one or more of the following apply:

   a) One or more of the arguments are implementor-defined.

   b) One or more of the arithmetic expressions that compose the equivalent arithmetic expression produce implementor-defined results.

c) The returned value is explicitly implementor-defined.

When a numeric or integer function does not have an equivalent arithmetic expression, its returned value is implementor-defined unless otherwise specified in the function definition.

15.5 Date and time conversion functions

The date conversion functions use the Gregorian calendar. The method for determining leap years is specified in ISO 8601:2004, 3.2.1, The Gregorian calendar.

15.5.1 Integer date form

The integer date form used by the date conversion functions is based on a starting date of Monday, January 1, 1601, which was chosen to establish a simple relationship between the integer date and DAY-OF-WEEK: integer date 1 was a Monday, DAY-OF-WEEK 1. A value in integer date form is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar. It shall be greater than zero and shall be less than or equal to the value of FUNCTION INTEGER-OF-DATE (99991231), which is 3,067,671.

15.5.2 Standard date form

The standard date form is YYYYMMDD, where YYYY represents the year, MM represents the month of that year, and DD represents the day of that month.

15.5.3 Julian date form

The Julian date form is YYYYDDD, where YYYY represents the year and DDD represents the ordinal date within that year.

15.5.4 Standard numeric time form

A value in standard numeric time form is a numeric value representing seconds past midnight.

If the LEAP-SECOND directive with the OFF phrase is in effect, the value shall be greater than or equal to zero and less than 86,400.

If the LEAP-SECOND directive with the ON phrase is in effect, the value shall be greater than or equal to zero and less than 86,401.

15.6 Summary of functions

Table 23, Table of functions, summarizes the functions that are available.

The 'arguments' column defines argument type and the 'type' column defines the type of the function, as follows:

Alph means alphabetic
Anum means alphanumeric
Bool means boolean
Ind means index
Int means integer
Key means a keyword
Loc means a locale
Nat means national
Num means numeric
Obj means object
Ord means an ordering table
Ptr means pointer
Type means a type declaration

Num in the arguments column includes Int. Both Int and Num are listed in the arguments column when the type of the argument determines the type of the function.

Anum in the arguments column includes strongly-typed group items. When the type of the argument determines the type of the function, the function is an alphanumeric function when any of the arguments is strongly-typed, even when all the arguments of the function are of the same type.

The 'Value returned' column gives a synopsis of the value returned; additional details are specified in the definition of the function.

Table 23 — Table of functions

Intrinsic-function-name	Arguments	Type	Value returned
ABS	Int1 or Num1	Depends upon argument	The absolute value of argument
ACOS	Num1	Num	Arccosine of Num1
ANNUITY	Num1, Int2	Num	Ratio of annuity paid for Int2 periods at interest of Num1 to initial investment of one
ASIN	Num1	Num	Arcsine of Num1
ATAN	Num1	Num	Arctangent of Num1
BOOLEAN-OF-INTEGER	Int1, Int2	Bool	A boolean item representing the binary value equivalent of the numeric value in argument-1
BYTE-LENGTH	Alph1 or Anum1 or Bool1 or Ind1 or Nat1 or Num1 or Obj 1 or Ptr1 or Type1	Int	Length of argument in number of bytes
CHAR	Int1	Anum	Character in position Int1 of the alphanumeric program collating sequence
CHAR-NATIONAL	Int1	Nat	Character in position Int1 of the national program collating sequence
COMBINED-DATETIME	Int1, Num2	Num	Numeric representation of combined integer date and standard numeric time
COS	Num1	Num	Cosine of Num1
CURRENT-DATE		Anum	Current date and time and local time differential
DATE-OF-INTEGER	Int1	Int	Standard date equivalent (YYYYMMDD) of integer date
DATE-TO-YYYYMMDD	Int1, Int2, Int3	Int	Argument-1 converted from YYMMDD to YYYYMMDD based on the values of argument-2 and argument-3
DAY-OF-INTEGER	Int1	Int	Julian date equivalent (YYYYDDD) of integer date

Table 23 — Table of functions (Continued)

Intrinsic-function-name	Arguments	Type	Value returned
DAY-TO-YYYYDDD	Int1, Int2, Int3	Int	Argument-1 converted from YYDDD to YYYYDDD based on the values of argument-2 and argument-3
DISPLAY-OF	Nat1, Anum2	Anum	Usage display representation of argument Nat1
E		Num	The value of *e*, the natural base
EXCEPTION-FILE		Anum	Information about the file exception that raised an exception
EXCEPTION-FILE-N		Nat	Information about the file exception that raised an exception
EXCEPTION-LOCATION		Anum	Implementor-defined location of statement causing an exception
EXCEPTION-LOCATION-N		Nat	Implementor-defined location of statement causing an exception
EXCEPTION-STATEMENT		Anum	Name of statement causing an exception
EXCEPTION-STATUS		Anum	Exception-name identifying last exception
EXP	Num1	Num	*e* raised to the power Num1
EXP10	Num1	Num	10 raised to the power Num1
FACTORIAL	Int1	Int	Factorial of Int1
FORMATTED-CURRENT-DATE	Anum1 or Nat1, Int2	Depends upon argument	Formatted date equivalent of current date and time
FORMATTED-DATE	Anum1 or Nat1, Int2	Depends upon argument	Formatted date equivalent of integer date contained in argument-2
FORMATTED-DATETIME	Anum1 or Nat1, Int2, Num3, Int4	Depends upon argument	Formatted date (from integer date in argument-2) and time (from standard numeric time in argument-3) in the format specified by argument-1. Offset from UTC, if the format requires it, is supplied by argument-4
FORMATTED-TIME	Anum1 or Nat1, Num2, Int3	Depends upon argument	Formatted time equivalent of standard numeric time contained in argument-2. Offset from UTC, if the format requires it, is supplied by argument-3
FRACTION-PART	Num1	Num	Fraction part of Num1
HIGHEST-ALGEBRAIC	Anum1 or Int1 or Nat1 or Num1	Int Num	Greatest algebraic value that may be represented in the argument
INTEGER	Num1	Int	The greatest integer not greater than Num1
INTEGER-OF-BOOLEAN	Bool1	Int	The numeric value of a BINARY-DOUBLE item whose bit configuration is the same as Bool1, right-justified.

Table 23 — Table of functions (Continued)

Intrinsic-function-name	Arguments	Type	Value returned
INTEGER-OF-DATE	Int1	Int	Integer date equivalent of standard date (YYYYMMDD)
INTEGER-OF-DAY	Int1	Int	Integer date equivalent of Julian date (YYYYDDD)
INTEGER-OF-FORMATTED-DATE	Anum1 or Nat1, Anum2 or Nat2	Int	Integer date equivalent of date contained in argument-2 whose format is described by argument-1
INTEGER-PART	Num1	Int	Integer part of Num1
LENGTH	Alph1 or Anum1 or Bool1 or Ind1 or Nat1 or Num1 or Obj1 or Ptr1 or Type1	Int	Length of argument in number of character positions or number of boolean positions
LOCALE-COMPARE	Alph1, Anum1, or Nat1 Alph2, Anum2, or Nat2 Loc3	Anum	A character indicating the result of comparing argument-1 to argument-2 using an ordering defined by a locale
LOCALE-DATE	Anum1 or Nat1, Loc2	Anum	A character string containing a date specified by argument-1 in a format specified by argument-2 and a locale
LOCALE-TIME	Anum1 or Nat1, Loc2	Anum	A character string containing a time specified by argument-1 in a format specified by a locale
LOCAL-TIME-FROM-SECONDS	Num1, Loc2	Anum	A character-string containing a time specified by argument-1, in a format specified by a locale specified by argument-2
LOG	Num1	Num	Natural logarithm of Num1
LOG10	Num1	Num	Logarithm to base 10 of Num1
LOWER-CASE	Alph1 or Anum1 or Nat1	Depends upon argument*	A character string with any uppercase letters in the argument set to lowercase
LOWEST-ALGEBRAIC	Anum1 or Int1 or Nat1 or Num1	Int Num	Lowest algebraic value that may be represented in the argument.
MAX	Alph1 ... or Anum1 ... or Ind1 ... or Int1 .... or Nat1 ... or Num1 ...	Depends upon arguments*	Value of maximum argument

Table 23 — Table of functions (Continued)

Intrinsic-function-name	Arguments	Type	Value returned
MEAN	Num1 ...	Num	Arithmetic mean of arguments
MEDIAN	Num1 ...	Num	Median of arguments
MIDRANGE	Num1 ...	Num	Mean of minimum and maximum arguments
MIN	Alph1 ... or Anum1 ... or Ind1 ... or Int1 ... or Nat1 ... or Num1 ...	Depends upon arguments*	Value of minimum argument
MOD	Int1, Int2	Int	Int1 modulo Int2
NATIONAL-OF	Anum1, Nat2	Nat	Usage national representation of argument Anum1
NUMVAL	Anum1 or Nat1	Num	Numeric value of simple numeric string
NUMVAL-C	Anum1 or Nat1, Anum2 or Nat2 or Key2 Loc2, Key3	Num	Numeric value of numeric string with optional commas and currency sign
NUMVAL-F	Anum1 or Nat1	Num	Numeric value of numeric string representing a floating-point number
ORD	Alph1 or Anum1 or Nat1	Int	Ordinal position of the argument in collating sequence
ORD-MAX	Alph1 ... or Anum1 ... or Ind1 .... or Nat1 ... or Num1 ...	Int	Ordinal position of maximum argument
ORD-MIN	Alph1 ... or Anum1 ... or Ind1 ... or Nat1 ... or Num1 ...	Int	Ordinal position of minimum argument
PI		Num	The value of π
PRESENT-VALUE	Num1, Num2 ...	Num	Present value of a series of future period-end amounts, Num2, at a discount rate of Num1
RANDOM	Int1	Num	Random number
RANGE	Int1 ... or Num1 ...	Depends upon argument	Value of maximum argument minus value of minimum argument
REM	Num1, Num2	Num	Remainder of Num1/Num2
REVERSE	Alph1 or Anum1 or Nat1	Depends upon argument*	Reverse order of the characters of the argument

Table 23 — Table of functions (Continued)

Intrinsic-function-name	Arguments	Type	Value returned
SECONDS-FROM-FORMATTED-TIME	Anum1 or Nat1, Anum2 or Nat2	Num	Standard numeric time equivalent of the data contained in argument-2 as described by the format described by argument-1
SECONDS-PAST-MIDNIGHT		Num	Seconds past midnight as provided by the system
SIGN	Num1	Int	The sign of Num1
SIN	Num1	Num	Sine of Num1
SQRT	Num1	Num	Square root of Num1
STANDARD-COMPARE	Alph1, Anum1, or Nat1 Alph2, Anum2, or Nat2 Ord3 Int4	Anum	A character indicating the result of comparing argument-1 to argument-2 using the ordering specified by argument-3 at the comparison level specified by argument-4
STANDARD-DEVIATION	Num1 ...	Num	Standard deviation of arguments
SUM	Int1 ... or Num1 ...	Depends upon arguments	Sum of arguments
TAN	Num1	Num	Tangent of Num1
TEST-DATE-YYYYMMDD	Int1	Int	0 if Int1 is a valid standard date; otherwise identifies the sub-field in error
TEST-DAY-YYYYDDD	Int1	Int	0 if Int1 is a valid Julian date; otherwise identifies the sub-field in error
TEST-FORMATTED-DATETIME	Anum1 or Nat1, Anum2 or Nat2	Int	0 if argument-2 conforms to the description in argument-1 and represents a valid date, time or combined representation according to that description; otherwise, identifies the character in error
TEST-NUMVAL	Anum1 or Nat1	Int	0 if argument-1 conforms to the requirements of the NUMVAL function; otherwise identifies the character in error
TEST-NUMVAL-C	Anum1 or Nat1, Anum2 or Nat2 or Key2 Loc2, Key3	Int	0 if argument-1 conforms to the requirements of the NUMVAL-C function; otherwise identifies the character in error
TEST-NUMVAL-F	Anum1 or Nat1	Int	0 if argument-1 conforms to the requirements of the NUMVAL-F function; otherwise identifies the character in error
TRIM	Anum1 or Nat1	Depends upon argument	The value of the argument with leading spaces, trailing spaces, or both, deleted.
UPPER-CASE	Alph1 or Anum1 or Nat1	Depends upon argument*	A character string with any lowercase letters in the argument set to uppercase

Table 23 — Table of functions (Continued)

Intrinsic-function-name	Arguments	Type	Value returned
VARIANCE	Num1 ...	Num	Variance of argument
WHEN-COMPILED		Anum	Date and time compilation unit was compiled
YEAR-TO-YYYY	Int1, Int2, Int3	Int	Argument-1 converted from YY to YYYY based on the values of argument-2 and argument-3
* A function that has only alphabetic arguments is type alphanumeric.			

15.7 ABS function

The ABS function returns the absolute value of the argument.

The type of this function depends on the argument type as follows:

Argument type	Function type
Integer	Integer
Numeric	Numeric

15.7.1 General format

FUNCTION ABS (argument-1)

15.7.2 Arguments

1) Argument-1 shall be of class numeric.

15.7.3 Returned values

1) The equivalent arithmetic expression is as follows:

   a) When the value of argument-1 is zero or positive,

      (argument-1)

   b) When the value of argument-1 is negative,

      (– (argument-1))

15.8 ACOS function

The ACOS function returns a numeric value in radians that approximates the arccosine of argument-1.

The type of this function is numeric.

15.8.1 General format

FUNCTION ACOS (argument-1)

15.8.2 Arguments

1) Argument-1 shall be of class numeric.

2) The value of argument-1 shall be greater than or equal to –1 and less than or equal to +1.

15.8.3 Returned values

1) The returned value is the approximation of the arccosine of argument-1 and is greater than or equal to zero and less than or equal to π.

15.9 ANNUITY function

The ANNUITY function (annuity immediate) returns a numeric value that approximates the ratio of an annuity paid at the end of each period for the number of periods specified by argument-2 to an initial investment of one. Interest is earned at the rate specified by argument-1 and is applied at the end of the period, before the payment.

The type of this function is numeric.

15.9.1 General format

FUNCTION ANNUITY (argument-1 argument-2)

15.9.2 Arguments

1) Argument-1 shall be of class numeric.

2) The value of argument-1 shall be greater than or equal to zero.

3) Argument-2 shall be a positive integer.

15.9.3 Returned values

1) The equivalent arithmetic expression is as follows:

   a) When the value of argument-1 is zero,

      (1 / (argument-2))

   b) When the value of argument-1 is not zero,

      (argument-1 / (1 – (1 + argument-1)** (– (argument-2))))

15.10 ASIN function

The ASIN function returns a numeric value in radians that approximates the arcsine of argument-1.

The type of this function is numeric.

15.10.1 General format

FUNCTION ASIN (argument-1)

15.10.2 Arguments

1) Argument-1 shall be of class numeric.

2) The value of argument-1 shall be greater than or equal to –1 and less than or equal to +1.

15.10.3 Returned values

1) The returned value is the approximation of the arcsine of argument-1 and is greater than or equal to $-\pi/2$ and less than or equal to $+\pi/2$.

15.11 ATAN function

The ATAN function returns a numeric value in radians that approximates the arctangent of argument-1.

The type of this function is numeric.

15.11.1 General format

FUNCTION ATAN (argument-1)

15.11.2 Arguments

1) Argument-1 shall be of class numeric.

15.11.3 Returned values

1) The returned value is the approximation of the arctangent of argument-1 and is greater than $-\pi/2$ and less than $+\pi/2$.

15.12 BOOLEAN-OF-INTEGER function

The BOOLEAN-OF-INTEGER function returns a boolean item of usage bit representing the binary value of argument-1. Argument-2 specifies the length of the boolean data item that is returned.

The function type is boolean.

15.12.1 General format

FUNCTION BOOLEAN-OF-INTEGER (argument-1 argument-2)

15.12.2 Arguments

1) Argument-1 shall be a positive integer.

2) Argument-2 shall be a positive nonzero integer.

15.12.3 Returned values

1) The returned value is a boolean item of usage bit that has the same bit configuration as the binary representation of the value of argument-1, where the rightmost boolean position is the low-order binary digit. The boolean value is zero-filled or truncated on the left, if necessary, in order to return a boolean item whose length is specified by argument-2 in terms of boolean positions.

   NOTE Binary representation is a mathematical concept. It is not required that this representation be the same as a COBOL representation.

15.13 BYTE-LENGTH function

The BYTE-LENGTH function returns an integer equal to the length of the argument in bytes.

The type of the function is integer.

15.13.1 General format

FUNCTION BYTE-LENGTH (argument-1 [PHYSICAL] }

15.13.2 Arguments

1) Argument-1 shall be an alphanumeric or national literal, a based entry, a type-name, or a data item of any class or category.

15.13.3 Returned values

1) The returned value is an integer that is the length of argument-1 in number of bytes.

2) If any data description entry subordinate to the data description entry of argument-1 is described with the DEPENDING phrase of the OCCURS clause, then

   a) if argument-1 is a based entry unassociated with actual data or is a type declaration, the length of argument-1 is determined in accordance with the rules of the OCCURS clause for a receiving data item; otherwise

   b) the length of argument-1 is determined in accordance with the rules of the OCCURS clause for a sending data item.

3) The returned value shall include the number of implicit filler positions, if any, in argument-1.

4) When argument-1 does not occupy an integral number of bytes, the returned value is rounded to the next larger integer value.

5) If argument-1 is an any-length elementary item, the current length of argument-1 in bytes is returned.

   NOTE Any prefixed fields or delimiter characters are not included in the current length.

6) If argument-1 is a variable-length group and the PHYSICAL argument is not specified, the value returned is the sum of the following:

   a) the lengths of all subordinate non-variable-length data-items;

   b) the lengths of all subordinate dynamic-capacity tables based on their current capacity;

   c) the current lengths of all subordinate any-length elementary items.

7) If argument-1 is a variable-length group and the PHYSICAL argument is specified, the returned value is the length of argument-1 in number of bytes.

   If argument-1 is not physically located where it is defined, the returned value includes only the length of the implementor-defined pointer. If argument-1 is physically located where it is defined, BYTE-LENGTH returns the same value same value that would be returned had the PHYSICAL argument not been specified.

15.14 CHAR function

The CHAR function returns a one-character alphanumeric value that is a character in the alphanumeric program collating sequence having the ordinal position equal to the value of argument-1.

The type of this function is alphanumeric.

15.14.1 General format

FUNCTION CHAR (argument-1)

15.14.2 Arguments

1) Argument-1 shall be an integer.

2) The value of argument-1 shall be greater than zero and less than or equal to the number of positions in the alphanumeric program collating sequence.

15.14.3 Returned values

1) The returned value shall be the character in the alphanumeric program collating sequence having the ordinal position specified by argument-1.

2) If more than one character has the same position in the alphanumeric program collating sequence, the character returned is the first character defined for that character position. If the order of multiple characters having the same position is undefined, the implementor shall define which of those multiple characters is returned; for a given implementation, collating sequence, and ordinal position, every invocation of the CHAR function shall return the same character.

15.15 CHAR-NATIONAL function

The CHAR-NATIONAL (national character) function returns a one-character value that is a character in the national program collating sequence having the ordinal position equal to the value of the argument.

The type of the function is national.

15.15.1 General format

FUNCTION CHAR-NATIONAL (argument-1)

15.15.2 Arguments

1) Argument-1 shall be an integer.

2) The value of argument-1 shall be greater than zero and less than or equal to the number of positions in the national program collating sequence.

15.15.3 Returned values

1) The returned value is the character in the national program collating sequence having the ordinal position specified by argument-1.

2) If more than one character has the same position in the national program collating sequence, the character returned is the first character defined for that character position. If the order of multiple characters having the same position is undefined, the implementor shall define which of those multiple characters is returned; for a given implementation, collating sequence, and ordinal position, every invocation of the CHAR-NATIONAL function shall return the same character.

15.16 COMBINED-DATETIME function

The COMBINED-DATETIME function combines a date in integer date form and a time in standard numeric time form into a single numeric item from which both date and time components can be derived.

The type of this function is numeric.

15.16.1 General format

FUNCTION COMBINED-DATETIME (argument-1 argument-2)

15.16.2 Arguments

1) Argument-1 shall be in integer date form.

2) Argument-2 shall be in standard numeric time form.

15.16.3 Returned values

1) The equivalent arithmetic expression is as follows:

argument-1 + (argument-2 / 100000)

15.17 COS function

The COS function returns a numeric value that approximates the cosine of an angle or arc, expressed in radians, that is specified by argument-1.

The type of this function is numeric.

15.17.1 General format

FUNCTION COS (argument-1)

15.17.2 Arguments

1) Argument-1 shall be of class numeric.

15.17.3 Returned values

1) The returned value is the approximation of the cosine of argument-1 and is greater than or equal to –1 and less than or equal to +1.

15.18 CURRENT-DATE function

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date, time of day, and local time differential factor provided by the system on which the function is evaluated.

The type of this function is alphanumeric.

15.18.1 General format

FUNCTION CURRENT-DATE

15.18.2 Returned values

1) The character positions returned, numbered from left to right, are:

Character Positions	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric characters of the seconds past the minute in the range: — 00 through 59 when a LEAP-SECOND directive with the OFF phrase is in effect — 00 through nn, where nn is defined by the implementor, when a LEAP-SECOND directive with the ON phrase is in effect.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the function is evaluated does not have the facility to provide the fractional part of a second.
17	Either the character '–', the character '+', or the character '0'. The character '–' is returned if the local time indicated in the previous character positions is behind Coordinated Universal Time. The character '+' is returned if the local time indicated is the same as or ahead of Coordinated Universal time. The character '0' is returned if the system on which this function is evaluated does not have the facility to provide the local time differential factor.
18-19	If character position 17 is '–', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the local time is behind Coordinated Universal Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the local time is ahead of Coordinated Universal Time. If character position 17 is '0', the value 00 is returned.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the local time is ahead of or behind Coordinated Universal Time, depending on whether character position 17 is '+' or '–', respectively. If character position 17 is '0', the value 00 is returned.

15.19 DATE-OF-INTEGER function

The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD).

The type of this function is integer.

15.19.1 General format

FUNCTION DATE-OF-INTEGER (argument-1)

15.19.2 Arguments

1) Argument-1 shall be a value in integer date form.

15.19.3 Returned values

1) The returned value represents the standard date equivalent of the integer specified in argument-1.

2) The returned value is in the form (YYYYMMDD) where YYYY represents a year in the Gregorian calendar; MM represents the month of that year; and DD represents the day of that month.

15.20 DATE-TO-YYYYMMDD function

The DATE-TO-YYYYMMDD function converts argument-1 from the form YYmmdd to the form YYYYmmdd. Argument-2, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding window, into which the year of argument-1 falls. Argument-3 specifies the year at the time of execution.

The type of the function is integer.

15.20.1 General format

FUNCTION DATE-TO-YYYYMMDD (argument-1 [argument-2 [argument-3]])

15.20.2 Arguments

1) Argument-1 shall be a positive integer less than 1000000.

   NOTE This function does not check argument -1 to ensure that it is a valid date. The returned value can be an argument to the TEST-DATE-YYYYMMDD function to check its validity.

2) Argument-2 shall be an integer.

3) If argument-2 is omitted, the function shall be evaluated as though 50 were specified for argument-2.

4) Argument-3 shall be an integer greater than 1600 and less than 10000.

5) If argument-3 is omitted, the function shall be evaluated as though the following were specified for argument-3:

   (FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4)))

6) The sum of the year at the time of execution and the value of argument-2 shall be less than 10000 and greater than 1699.

15.20.3 Returned values

1) The equivalent arithmetic expression is as follows:

   (FUNCTION YEAR-TO-YYYY (YY, argument-2, argument-3) * 10000 + mmdd)

   where

   YY = FUNCTION INTEGER (argument-1/10000)
   mmdd = FUNCTION MOD (argument-1, 10000)

   and where argument-1, argument-2 and argument-3 are the same as argument-1, argument-2, and argument-3 of the DATE-TO-YYYYMMDD function reference itself.

   NOTE 1 In the year 2002 the returned value for FUNCTION DATE-TO-YYYYMMDD (851003, 10) is 19851003. In the year 1994 the returned value for FUNCTION DATE-TO-YYYYMMDD (981002, (–10)) is 18981002.

   NOTE 2 See the notes for the YEAR-TO-YYYY function for a discussion of how to specify a fixed window or a sliding window algorithm.

15.21 DAY-OF-INTEGER function

The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD).

The type of this function is integer.

15.21.1 General format

FUNCTION DAY-OF-INTEGER (argument-1)

15.21.2 Arguments

1) Argument-1 shall be a value in integer date form.

15.21.3 Returned values

1) The returned value represents the Julian equivalent of the integer specified in argument-1.

2) The returned value is an integer of the form (YYYYDDD) where YYYY represents a year in the Gregorian calendar and DDD represents the day of that year.

15.22 DAY-TO-YYYYDDD function

The DAY-TO-YYYYDDD function converts argument-1 from the form YYnnn to the form YYYYnnn. Argument-2, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding window, into which the year of argument-1 falls. Argument-3 specifies the year at the time of execution.

The type of the function is integer.

15.22.1 General format

FUNCTION DAY-TO-YYYYDDD (argument-1 [argument-2 [argument-3]])

15.22.2 Arguments

1) Argument-1 shall be a positive integer less than 100000.

   NOTE This function does not check argument -1 to ensure that it is a valid date. The returned value can be an argument to the TEST-DAY-YYYYDDD function to check its validity.

2) Argument-2 shall be an integer.

3) If argument-2 is omitted, the function shall be evaluated as though 50 were specified for argument-2.

4) Argument-3 shall be an integer greater than 1600 and less than 10000.

5) If argument-3 is omitted, the function shall be evaluated as though the following were specified for argument-3:

   (FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4)))

6) The sum of the values of argument-2 and argument-3 shall be less than 10000 and greater than 1699.

15.22.3 Returned values

1) The equivalent arithmetic expression is as follows:

   (FUNCTION YEAR-TO-YYYY (YY, argument-2, argument-3) * 1000 + nnn)

   where

   YY = FUNCTION INTEGER (argument-1/1000)
   nnn = FUNCTION MOD (argument-1, 1000)

   and where argument-1, argument-2 and argument-3 are the same as argument-1, argument-2, and argument-3 of the DAY-TO-YYYYDDD function reference itself.

   NOTE 1 In the year 2002 the returned value for FUNCTION DAY-TO-YYYYDDD (10004, 20) is 2010004. In the year 2013 the returned value for FUNCTION DAY-TO-YYYYDDD (95005, (–10)) is 1995005.

   NOTE 2 See the notes for the YEAR-TO-YYYY function for a discussion of how to specify a fixed window or a sliding window algorithm.

15.23 DISPLAY-OF function

The DISPLAY-OF function returns a character string containing the alphanumeric coded character set representation of the national characters in the argument.

The type of the function is alphanumeric.

15.23.1 General format

FUNCTION DISPLAY-OF (argument-1 [argument-2])

15.23.2 Arguments

1) Argument-1 shall be of class national.

2) Argument-2 shall be of class alphabetic or alphanumeric and shall be one character position in length. Argument-2 specifies an alphanumeric substitution character for use in conversion of national characters for which there is no corresponding alphanumeric character.

15.23.3 Returned values

1) A character string is returned with each national character of argument-1 converted to its corresponding alphanumeric character representation, if any. The implementor shall define the correspondence of characters between the alphanumeric character set and the national character set for purposes of conversion with the DISPLAY-OF function.

2) If argument-2 is specified, the alphanumeric substitution character is returned for each national character in argument-1 that has no corresponding alphanumeric character representation.

3) If argument-2 is unspecified and argument-1 contains a national character for which there is no corresponding alphanumeric character representation, an implementor-defined substitution character is used as the corresponding alphanumeric character and the EC-DATA-CONVERSION exception condition is set to exist.

4) The length of the returned value is the number of character positions of usage display required to hold the converted argument and depends on the number of characters contained in argument-1.

15.24 E function

The E function returns an approximation of *e*, the base of natural logarithms.

The type of the function is numeric.

15.24.1 General format

FUNCTION E

15.24.2 Returned values

1) If native arithmetic is in effect, the returned value is an implementor-defined approximation of the arithmetic expression

   (2 + 0.71828182845904523536028747135 26).

2) If standard arithmetic is in effect, the equivalent arithmetic expression is

   (2 + 0.71828182845904523536029747135 26)

3) If standard-binary arithmetic is in effect, the returned value is the exact value of the arithmetic expression

   (14,114,126,198,520,207,781,233,383,725,636,853 / (2 ** 112))

   in Binary128 format, normalized as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings.

4) If standard-decimal arithmetic is in effect, the equivalent arithmetic expression is the exact value of the arithmetic expression

   (2.71828182845904523536028747135 2662)

   in decimal128 format with decimal encoding of the significand, as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.5, Decimal interchange format encodings.

15.25 EXCEPTION-FILE function

The EXCEPTION-FILE function returns an alphanumeric character string that is the I-O status value and file-name of the file connector, if any, associated with the last exception status.

The type of the function is alphanumeric.

15.25.1 General Format

FUNCTION EXCEPTION-FILE

15.25.2 Returned values

1) The returned value is an alphanumeric character string that has a length that is based on its contents and the contents are as follows:

   a) If the last exception status is not an EC-I-O exception condition, the returned value is two alphanumeric zeros.

   b) The returned value is two alphanumeric spaces when the last exception status indicates an EC-I-O exception condition that originates from one of the following statements:

      — a RAISE statement

      — an EXIT statement with a RAISING phrase that specifies an EC-I-O exception-name

      — a GOBACK statement with a RAISING phrase that specifies an EC-I-O exception-name.

   c) Otherwise, the returned value is a character string that is as long as is needed to contain the I-O status value and the file-name. The first two characters are the I-O status value in alphanumeric characters. The succeeding characters contain the file-name exactly as specified in the SELECT clause converted at runtime to the runtime alphanumeric character set.

15.26 EXCEPTION-FILE-N function

The EXCEPTION-FILE-N function returns a national character string that is the I-O status value and file-name of the file connector, if any, associated with the last exception status.

The type of the function is national.

15.26.1 General Format

FUNCTION EXCEPTION-FILE-N

15.26.2 Returned values

1) The returned value is a national character string that has a length that is based on its contents and the contents are as follows:

   a) If the last exception status is not an EC-I-O exception condition, the returned value is two national zeros.

   b) The returned value is two national spaces when the last exception status indicates an EC-I-O exception condition that originates from one of the following statements:

      — a RAISE statement

      — an EXIT statement with a RAISING phrase that specifies an EC-I-O exception-name

      — a GOBACK statement with a RAISING phrase that specifies an EC-I-O exception-name.

   c) Otherwise, the returned value is a character string that is as long as is needed to contain the I-O status value and the file-name. The first two characters are the I-O status value in national characters. The succeeding characters contain the file-name exactly as specified in the SELECT clause converted at runtime to the runtime national character set.

15.27 EXCEPTION-LOCATION function

The EXCEPTION-LOCATION function returns an alphanumeric character string that is the implementor-defined location of the statement associated with the last exception status.

The type of the function is alphanumeric.

15.27.1 General format

FUNCTION EXCEPTION-LOCATION

15.27.2 Returned values

1) If the LOCATION option of the TURN directive that enabled checking for the exception condition associated with the last exception status is not specified and the implementor does not save the location information, the returned value is one alphanumeric space character.

2) If the LOCATION option of the associated TURN directive was specified, the returned value is an alphanumeric character string that has a length that is based on its contents and the contents are as follows:

    a) If the last exception status indicates that no exception condition was raised, the returned value is one alphanumeric space character.

    b) Otherwise, the returned value is a character string that is as long as is needed to contain the location information. All of the names are exactly as specified in the source element containing the statement and they are converted at runtime to the runtime alphanumeric character set. The character string is composed of three parts as follows:

        1. The name of the runtime element as specified in the FUNCTION-ID, METHOD-ID, or PROGRAM-ID paragraph of the function, method, or program containing the statement. In the case of a propagated exception condition, the name is that of the function, method, or program in which the exception condition that was propagated actually occurred. The name is immediately followed by a semicolon and a space character.

        2. The procedure-name of the procedure that contains the statement as follows:

            a. If there is no paragraph-name or section-name in the source element, a semicolon and a space character are appended.

            b. If there is a paragraph-name, the paragraph-name is appended and, if the paragraph is within a section, the section-name of the section containing the paragraph is appended prefixed by the alphanumeric characters ' OF '. This is followed by a semicolon and a space character.

            c. If there is a section-name and no paragraph-name, the section-name is appended followed by a semicolon and a space character.

        3. An implementor-defined identifier of the source line that contains the beginning of the statement is then appended.

            NOTE The user cannot rely on the value returned being consistent from one compilation of the same compilation unit to the next. Therefore, it should not be used in a comparison with a known value.

15.28 EXCEPTION-LOCATION-N function

The EXCEPTION-LOCATION-N function returns a national character string that is the implementor-defined location of the statement associated with the last exception status.

The type of the function is national.

15.28.1 General format

FUNCTION EXCEPTION-LOCATION-N

15.28.2 Returned values

1) If the LOCATION option of the TURN directive that enabled checking for the exception condition associated with the last exception status is not specified and the implementor does not save the location information, the returned value is one national space character.

2) If the LOCATION option of the associated TURN directive was specified, the returned value is a national character string that has a length that is based on its contents and the contents are as follows:

    a) If the last exception status indicates that no exception condition was raised, the returned value is one national space character.

    b) Otherwise, the returned value is a character string that is as long as is needed to contain the location information. All of the names are exactly as specified in the source element containing the statement and they are converted at runtime to the runtime national character set. The character string is composed of three parts as follows:

        1. The name of the runtime element as specified in the FUNCTION-ID, METHOD-ID, or PROGRAM-ID paragraph of the function, method, or program containing the statement. In the case of a propagated exception condition, the name is that of the function, method, or program in which the exception condition that was propagated actually occurred. The name is immediately followed by a semicolon and a national space character.

        2. The procedure-name of the procedure that contains the statement as follows:

            a. If there is no paragraph-name or section-name in the source element, a semicolon and a space character are appended.

            b. If there is a paragraph-name, the paragraph-name is appended and, if the paragraph is within a section, the section-name of the section containing the paragraph is appended prefixed by the national characters ' OF '. This is followed by a semicolon and a space character.

            c. If there is a section-name and no paragraph-name, the section-name is appended followed by a semicolon and a space character.

        3. An implementor-defined identifier of the source line that contains the beginning of the statement is then appended.

    NOTE The user cannot rely on the value returned being consistent from one compilation of the same compilation unit to the next. Therefore, it should not be used in a comparison with a known value.

15.29 EXCEPTION-STATEMENT function

The EXCEPTION-STATEMENT function returns an alphanumeric value that is the name of the statement that caused the associated exception condition.

The type of the function is alphanumeric.

15.29.1 General format

FUNCTION EXCEPTION-STATEMENT

15.29.2 Returned values

1) If the LOCATION option of the TURN directive that enabled checking for the exception condition associated with the last exception status is not specified and the implementor does not save the location information, the returned value is 31 spaces.

2) If the LOCATION option of the associated TURN directive was specified, the returned value is a 31-character alphanumeric character-string that is the name of the statement that caused the exception condition to be raised in uppercase letters, left-justified and space-filled on the right.

3) The names of the statements are given in table 13, Procedural statements, in the column labeled 'Statement name'.

15.30 EXCEPTION-STATUS function

The EXCEPTION-STATUS function returns an alphanumeric value that is the exception-name associated with the last exception status.

The type of the function is alphanumeric.

15.30.1 General format

FUNCTION EXCEPTION-STATUS

15.30.2 Returned values

1) A 31-character, left-justified, alphanumeric character string that is the exception-name or the value 'EXCEPTION-OBJECT', as applicable, associated with the last exception status. All letters in the exception-name are returned as uppercase letters and all unused characters are alphanumeric spaces. If the last exception status indicates no exception, alphanumeric spaces are returned.

15.31 EXP function

The EXP function returns an approximation of the value of *e* raised to the power of the argument.

The type of the function is numeric.

15.31.1 General format

FUNCTION EXP (argument-1)

15.31.2 Arguments

1) Argument-1 shall be of class numeric.

15.31.3 Returned values

1) The equivalent arithmetic expression is:

(FUNCTION E ** (argument-1))

15.32 EXP10 function

The EXP10 function returns an approximation of the value of 10 raised to the power of the argument.

The type of the function is numeric.

15.32.1 General format

FUNCTION EXP10 (argument-1)

15.32.2 Arguments

1) Argument-1 shall be of class numeric.

15.32.3 Returned values

1) The equivalent arithmetic expression is:

(10 ** (argument-1))

15.33 FACTORIAL function

The FACTORIAL function returns an integer that is the factorial of argument-1.

The type of this function is integer.

15.33.1 General format

FUNCTION FACTORIAL (argument-1)

15.33.2 Arguments

1) Argument-1 shall be an integer greater than or equal to zero.

15.33.3 Returned values

1) The equivalent arithmetic expression is as follows:

   a) When the value of argument-1 is 0 or 1,

      (1)

   b) When the value of argument-1 is 2,

      (2)

   c) When the value of argument-1 is n,

      (n * (n – 1) * (n – 2) * ... * 1)

15.34 FORMATTED-CURRENT-DATE function

The FORMATTED-CURRENT-DATE function returns a character string representing the current date and time provided by the system on which the function is evaluated. The content of the returned value is formatted according to the format in the argument.

The type of this function depends on the argument type as follows:

Argument type	Function type
Alphanumeric	Alphanumeric
National	National

15.34.1 General format

FUNCTION FORMATTED-CURRENT-DATE (argument-1)

15.34.2 Arguments

1) Argument-1 shall be a national or alphanumeric literal.

2) The content of argument-1 shall be a combined date and time format.

15.34.3 Returned values

1) The returned value is a representation of the current date and time provided by the system on which the function is evaluated. The returned value is formatted according to the format in argument-1.

2) The implementor shall define the accuracy of the portion of the returned value that corresponds to the time format portion of the argument.

15.35 FORMATTED-DATE function

The FORMATTED-DATE function uses a format to convert a date in integer date form to a date in the requested format.

The type of this function depends on the type of argument-1 as follows:

Argument type	Function type
Alphanumeric	Alphanumeric
National	National

15.35.1 General format

FUNCTION FORMATTED-DATE (argument-1 argument-2)

15.35.2 Arguments

1) Argument-1 shall be a national or alphanumeric literal.

2) The content of argument-1 shall be a date format.

3) Argument-2 shall be a value in integer date form.

15.35.3 Returned values

1) The returned value is a representation of the date contained in argument-2 according to the format in argument-1.

15.36 FORMATTED-DATETIME function

The FORMATTED-DATETIME function uses a combined time and date format to convert and combine a date in integer date form and a numeric time expressed as seconds past midnight to a produce a formatted date and time representation according to that combined date and time format.

The type of this function depends on the type of argument-1 as follows:

Argument type	**Function type**
Alphanumeric	Alphanumeric
National	National

15.36.1 General format

FUNCTION FORMATTED-DATETIME (argument-1 argument-2 argument-3 [argument-4])

15.36.2 Arguments

1) Argument-1 shall be a national or alphanumeric literal.

2) The content of argument-1 shall be a combined date and time format.

3) Argument-2 shall be a value in integer date form.

4) Argument-3 shall be a value in standard numeric time form.

   NOTE The offset value 1439 represents 23 hours 59 minutes, which is one minute less than a day.

5) Argument-4 is an integer specifying the offset from UTC expressed in minutes. If argument-4 is specified, the magnitude of the value shall be less than or equal to 1439.

6) Argument-4 shall not be specified if the time portion of the format in argument-1 is neither a UTC format nor an offset format.

7) If argument-4 is omitted and the time portion of the format in argument-1 is a UTC format or an offset format, the function shall be evaluated as though 0 were specified for argument-4.

15.36.3 Returned values

1) The returned value is a representation of the date contained in argument-2 combined with the time contained in argument-3 according to the format in argument-1.

2) If the format in argument-1 indicates that the returned value is to be expressed in UTC, the time portion of the returned value reflects the adjustment of the value in argument-3 by the offset in argument-4.

3) If the format in argument-1 indicates that the time is to be returned as an offset from UTC, the value in argument-3 is reflected directly in the time portion of the returned value, and the offset in argument-4 is reflected directly in the offset portion of the returned value.

15.37 FORMATTED-TIME function

The FORMATTED-TIME function uses a format to convert a value representing seconds past midnight to a formatted time of day in the requested format.

The type of this function depends on the type of argument-1 as follows:

Argument type	Function type
Alphanumeric	Alphanumeric
National	National

15.37.1 General format

FUNCTION FORMATTED-TIME (argument-1 argument-2 [argument-3])

15.37.2 Arguments

1) Argument-1 shall be a national or alphanumeric literal.

2) The content of argument-1 shall be a time format.

3) Argument-2 shall be a value in standard numeric time form.

4) Argument-3 is an integer representation of offset from UTC expressed in minutes. If argument-3 is specified, the magnitude of the value shall be less than or equal to 1439.

   NOTE The offset value 1439 represents 23 hours 59 minutes, which is one minute less than a day.

5) Argument-3 shall not be specified if the time portion of the format in argument-1 is neither a UTC format nor an offset format.

6) If argument-3 is omitted and the time portion of the format in argument-1 is a UTC format or an offset format, the function shall be evaluated as though 0 were specified for argument-3.

15.37.3 Returned values

1) The returned value is a representation of the standard numeric time contained in argument-2 according to the format in argument-1.

2) If the format in argument-1 indicates that the returned value is to be expressed in UTC, the time portion of the returned value reflects the adjustment of the value in argument-2 by the offset in argument-3.

3) If the format in argument-1 indicates that the time is to be returned as an offset from UTC, the value in argument-2 is reflected directly in the time portion of the returned value, and the offset in argument-3 is reflected directly in the offset portion of the returned value.

15.38 FRACTION-PART function

The FRACTION-PART function returns a numeric value that is the fraction portion of the argument.

The type of the function is numeric.

15.38.1 General format

FUNCTION FRACTION-PART (argument-1)

15.38.2 Arguments

1) Argument-1 shall be of the class numeric.

15.38.3 Returned values

1) The equivalent arithmetic expression is:

(argument-1 – FUNCTION INTEGER-PART (argument-1))

where the argument for the INTEGER-PART function is the same as for the FRACTION-PART function itself.

NOTE If the value of argument-1 is +1.5, +0.5 is returned. If the value of argument-1 is –1.5, –0.5 is returned.

15.39 HIGHEST-ALGEBRAIC function

The HIGHEST-ALGEBRAIC function returns a value that is equal to the greatest algebraic value that may be represented in argument-1.

The type of this function depends upon the argument types as follows:

Argument type	**Function type**
Alphanumeric	Numeric
Integer	Integer
National	Numeric
Numeric	Numeric

15.39.1 General format

FUNCTION HIGHEST-ALGEBRAIC (argument-1)

15.39.2 Arguments

1) Argument-1 shall be a data item of category numeric or numeric-edited and shall not be an integer or numeric function.

15.39.3 Returned values

1) The value returned is equal to the positive algebraic value of greatest finite magnitude that may be represented in argument-1.

NOTE The following illustrates the expected results for some values of argument-1.

Argument-1 characteristics	**Value returned**
S999	+999
S9(4) BINARY	+9999
99V9(3)	+99.999
$**,**9.99BCR	+99999.99
$**,**9.99	+99999.99
BINARY-CHAR SIGNED	+127 (assuming an 8-bit representation)
BINARY-CHAR UNSIGNED	+255 (assuming an 8-bit representation)

15.40 INTEGER function

The INTEGER function returns the greatest integer value that is less than or equal to the argument.

The type of this function is integer.

15.40.1 General format

FUNCTION INTEGER (argument-1)

15.40.2 Arguments

1) Argument-1 shall be of class numeric.

15.40.3 Returned values

1) The returned value is the greatest integer less than or equal to the value of argument-1.

NOTE For example:

- If the value of argument-1 is -1.5, the value -2 is returned.
- If the value of argument-1 is +1.5, the value +1 is returned.
- If the value of argument-1 is zero, the value zero is returned.

A statement of the form

COMPUTE X ROUNDED MODE IS TRUNCATION = INTEGER (Y)

produces results analogous to those produced by a statement of the form

COMP[UTE X ROUNDED MODE IS TOWARD-LESSER = Y.

presuming that X is an integer data item.

The INTEGER-PART function is similar but returns different values for negative numbers.

15.41 INTEGER-OF-BOOLEAN function

The INTEGER-OF-BOOLEAN function returns the numeric value of the boolean string in argument-1.

The function type is integer.

15.41.1 General format

FUNCTION INTEGER-OF-BOOLEAN (argument-1)

15.41.2 Arguments

1) Argument-1 shall be of class boolean.

15.41.3 Returned values

1) The returned value is determined as follows:

   a) Argument-1 is assigned to a temporary boolean data item of usage bit described with the same number of boolean positions as argument-1.

   b) The unsigned binary value represented by the same bit configuration as the bit configuration of that temporary boolean data item is determined.

      NOTE Binary representation is a mathematical concept. It is not required that this representation be the same as a COBOL representation.

   c) The numeric value determined in subrule 1b is the returned value.

15.42 INTEGER-OF-DATE function

The INTEGER-OF-DATE function converts a date in the Gregorian calendar from standard date form (YYYYMMDD) to integer date form.

The type of this function is integer.

15.42.1 General format

FUNCTION INTEGER-OF-DATE (argument-1)

15.42.2 Arguments

1) Argument-1 shall be an integer of the form YYYYMMDD, whose value is obtained from the calculation (YYYY * 10,000) + (MM * 100) + DD.

   a) YYYY represents the year in the Gregorian calendar. It shall be an integer greater than 1600 and less than 10000.

   b) MM represents a month and shall be a positive integer less than 13.

   c) DD represents a day and shall be a positive integer less than 32 provided that it is valid for the specified month and year combination.

15.42.3 Returned values

1) The returned value is in integer date form.

15.43 INTEGER-OF-DAY function

The INTEGER-OF-DAY function converts a date in the Gregorian calendar from Julian date form (YYYYDDD) to integer date form.

The type of this function is integer.

15.43.1 General format

FUNCTION INTEGER-OF-DAY (argument-1)

15.43.2 Arguments

1) Argument-1 shall be an integer of the form YYYYDDD, whose value is obtained from the calculation (YYYY * 1000) + DDD.

   a) YYYY represents the year in the Gregorian calendar. It shall be an integer greater than 1600 and less than 10000.

   b) DDD represents the day of the year. It shall be a positive integer less than 367 provided that it is valid for the year specified.

15.43.3 Returned values

1) The returned value is in integer date form.

15.44 INTEGER-OF-FORMATTED-DATE function

The INTEGER-OF-FORMATTED-DATE function converts a date that is in a specified format to integer date form.

The type of this function is integer.

15.44.1 General format

FUNCTION INTEGER-OF-FORMATTED-DATE (argument-1 argument-2)

15.44.2 Arguments

1) Argument-1 shall be a national or alphanumeric literal.

2) The content of argument-1 shall be either a date format or a combined date and time format.

3) Argument-2 shall be a data item of the same type as argument-1.

4) If argument-1 is a date format, the contents of argument-2 shall be a valid date in that format.

5) If argument-1 is a combined date and time format, the contents of argument-2 shall be a valid combined date and time in that format.

15.44.3 Returned values

1) The returned value is the integer date form equivalent of the date represented by argument-2 when analyzed according to argument-1.

NOTE If argument-1 contains a combined date and time format, the time portion of argument-2 is validated against the format in argument-1, but if valid does not impact the returned value.

15.45 INTEGER-PART function

The INTEGER-PART function returns an integer that is the integer portion of argument-1.

The type of this function is integer.

15.45.1 General format

FUNCTION INTEGER-PART (argument-1)

15.45.2 Arguments

1) Argument-1 shall be of class numeric.

15.45.3 Returned values

1) The equivalent arithmetic expression is:

(FUNCTION SIGN (argument-1) * FUNCTION INTEGER (FUNCTION ABS (argument-1)))

where the arguments for the SIGN and ABS functions are the same as for the INTEGER-PART function itself.

NOTE For example:

- If the value of argument-1 is -1.5, the value -1 is returned.
- If the value of argument-1 is +1.5, the value +1 is returned.
- If the value of argument-1 is zero, the value zero is returned.
- If the value of argument-1 is -1.0, the value -1 is returned.
- If the value of argument-1 is +1.0, the value +1 is returned.

The INTEGER function is similar but returns different values for negative numbers.

15.46 LENGTH function

The LENGTH function returns an integer equal to the length of the argument in alphanumeric character positions, national character positions, or boolean positions, depending on the class of the argument.

The type of this function is integer.

15.46.1 General format

FUNCTION LENGTH { argument-1 [PHYSICAL] }

15.46.2 Arguments

1) Argument-1 shall be an alphanumeric, national, or boolean literal; a data item of any class or category; a based entry; or a type-name.

15.46.3 Returned values

1) If argument-1 is a bit group item, an elementary boolean data item, a boolean literal, or a type declaration for a boolean item, the returned value is an integer equal to the length of argument-1 in boolean positions.

2) If argument-1 is a national group item, an elementary data item of usage national other than a boolean data item, a national literal, or a type declaration for a data item of usage national, the returned value is an integer equal to the length of argument-1 in national character positions.

3) If argument-1 is other than category boolean or usage national, the returned value is an integer equal to the length of argument-1 in alphanumeric character positions.

4) If any data description entry subordinate to the data description entry of argument-1 is described with the DEPENDING phrase of the OCCURS clause, then

   a) If argument-1 is a based item unassociated with actual data or is a type declaration, the length of argument-1 is determined in accordance with the rules of the OCCURS clause for a receiving data item, otherwise

   b) the length of argument-1 is determined in accordance with the rules of the OCCURS clause for a sending data item.

5) The returned length shall include the number of implicit FILLER positions, if any, in argument-1.

6) When the returned value is expressed as a number of alphanumeric character positions and argument-1 does not occupy an integral number of positions, the returned value is rounded to the next larger integer value.

7) If argument-1 is an any-length elementary item, the current length of argument-1 in bytes is returned.

   NOTE Any prefixed fields or delimiter characters are not included in the current length.

8) If argument-1 is a variable-length group and the PHYSICAL argument is not specified, the value returned is the sum of the following:

   a) the lengths of all subordinate non-variable-length data-items;

   b) the lengths of all subordinate dynamic-capacity tables based on their current capacity;

   c) the current lengths of all subordinate any-length elementary items.

9) If argument-1 is a variable-length group and the PHYSICAL argument is specified, the returned value the length of argument-1 in number of characters.

   If argument-1 is not physically located where it is defined, the returned value includes only the length of the implementor-defined pointer. If argument-1 is physically located where it is defined, LENGTH returns the same value that would be returned had the PHYSICAL argument not been specified.

15.47 LOCALE-COMPARE function

The LOCALE-COMPARE function returns a character indicating the result of comparing argument-1 and argument-2 using a culturally-preferred ordering defined by a locale.

The function type is alphanumeric.

15.47.1 General format

FUNCTION LOCALE-COMPARE (argument-1 argument-2 [locale-name-1])

15.47.2 Arguments

1) Argument-1 shall be of class alphabetic, alphanumeric, or national.

2) Argument-2 shall be of class alphabetic, alphanumeric, or national.

3) Arguments may be of different classes.

4) Locale-name-1 shall be associated with a locale in the SPECIAL-NAMES paragraph.

15.47.3 Returned values

1) If the arguments are of different classes, and one is national, the other argument is converted to class national for purposes of comparison.

2) For purposes of comparison, trailing spaces are truncated from the operands except that an operand consisting of all spaces is truncated to a single space.

3) If locale-name-1 is specified, the locale used for comparison is the one associated with locale-name-1; otherwise, the current locale is used. If the locale associated with locale-name-1 is not available, the EC-LOCALE-MISSING exception condition is set to exist.

4) Argument-1 and argument-2 are compared using the cultural ordering defined by the locale being used.

   NOTE Locale-based ordering is not necessarily a character-by-character comparison.

5) The returned value is:

   '=' if the arguments compare equal,
   '<' if argument-1 is less than argument-2,
   '>' if argument-1 is greater than argument-2.

6) The length of the returned value is 1.

15.48 LOCALE-DATE function

The LOCALE-DATE function returns a character string containing a date in a culturally-appropriate format specified by a locale.

The function type is alphanumeric.

15.48.1 General format

FUNCTION LOCALE-DATE (argument-1 [locale-name-1])

15.48.2 Arguments

1) Argument-1 shall be of class alphanumeric or national and shall be 8 character positions in length.

2) The content of argument-1 shall be a date in the same format as the year, month, and day returned in character positions 1 through 8 by the CURRENT-DATE function and shall be valid according to the definition of a returned value from that function.

3) Locale-name-1 shall be associated with a locale in the SPECIAL-NAMES paragraph.

15.48.3 Returned values

1) If locale-name-1 is specified, the locale used for formatting the date is the one associated with locale-name-1; otherwise, the current locale is used. If the locale associated with locale-name-1 is not available, the EC-LOCALE-MISSING exception condition is set to exist.

2) The returned value is a character-string containing the date specified by argument-1 in the appropriate date format as indicated in the locale by locale field d_fmt.

3) The length of the returned value depends on the format indicated in the locale.

15.49 LOCALE-TIME function

The LOCALE-TIME function returns a character string containing a time in a culturally-appropriate format specified by a locale.

The function type is alphanumeric.

15.49.1 General format

FUNCTION LOCALE-TIME (argument-1 [locale-name-1])

15.49.2 Arguments

1) Argument-1 shall be of class alphanumeric or national and shall be 6 character positions in length.

2) The content of argument-1 shall be in the same format as the hours, minutes, and seconds returned in character positions 9 through 14 of the CURRENT-DATE function and shall be valid according to the definition of a returned value from that function.

3) The content of argument-1 shall be valid according to the definition of a returned value from the CURRENT-DATE function, with the following exceptions:

   a) The hours past midnight shall be 00 through 24.

   b) The seconds past the minutes shall be 00 through 99.

      NOTE The value of seconds is not precisely specified because the maximum number of leap seconds that can appear in user data is not constrained. A LEAP-SECOND directive with the ON phrase need not be in effect in order for user data to contain leap seconds.

4) Locale-name-1 shall be associated with a locale in the SPECIAL-NAMES paragraph.

15.49.3 Returned values

1) If locale-name-1 is specified, the locale used for formatting the time is the one associated with locale-name-1; otherwise, the current locale is used. If the locale associated with locale-name-1 is not available, the EC-LOCALE-MISSING exception condition is set to exist.

2) The returned value is a character-string containing hours, minutes, and seconds of the time specified by argument-1 in the culturally-appropriate format indicated in the locale by locale field t_fmt.

3) The length of the returned value depends on the format indicated in the locale.

15.50 LOCALE-TIME-FROM-SECONDS function

The LOCALE-TIME-FROM-SECONDS function accepts a value in standard numeric time form and returns a character-string containing a time in a culturally-appropriate format specified by a locale.

The function type is alphanumeric.

15.50.1 General format

FUNCTION LOCALE-TIME-FROM-SECONDS (argument-1 [locale-name-1])

15.50.2 Arguments

1) Argument-1 shall be a numeric value in standard numeric time form.

2) Locale-name-1 shall be associated with a locale in the SPECIAL-NAMES paragraph.

15.50.3 Returned values

1) If locale-name-1 is specified, the locale used for formatting the time is the one associated with locale-name-1; otherwise, the current locale is used. If the locale associated with locale-name-1 is not available, the EC-LOCALE-MISSING exception condition is set to exist.

2) The returned value is a character-string containing hours, minutes, and seconds of the time specified by argument-1 in the culturally-appropriate format indicated in the locale by locale field t_fmt.

3) The length of the returned value depends on the format indicated in the locale.

15.51 LOG function

The LOG function returns a numeric value that approximates the logarithm to the base *e* (natural log) of argument-1.

The type of this function is numeric.

15.51.1 General format

FUNCTION LOG (argument-1)

15.51.2 Arguments

1) Argument-1 shall be of class numeric.

2) The value of argument-1 shall be greater than zero.

15.51.3 Returned values

1) The returned value is the approximation of the logarithm to the base *e* of argument-1.

15.52 LOG10 function

The LOG10 function returns a numeric value that approximates the logarithm to the base 10 of argument-1.

The type of this function is numeric.

15.52.1 General format

FUNCTION LOG10 (argument-1)

15.52.2 Arguments

1) Argument-1 shall be of class numeric.

2) The value of argument-1 shall be greater than zero.

15.52.3 Returned values

1) The returned value is the approximation of the logarithm to the base 10 of argument-1.

15.53 LOWER-CASE function

The LOWER-CASE function returns a character string that contains the value of argument-1 with any uppercase letters replaced by their corresponding lowercase letters.

The type of the function depends on the argument type as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
National	National

15.53.1 General format

FUNCTION LOWER-CASE (argument-1 [LOCALE locale-name-1])

15.53.2 Arguments

1) Argument-1 shall be of class alphabetic, alphanumeric, or national and shall be at least one character position in length.

15.53.3 Returned values

1) A character string with the content of argument-1 is returned, with any uppercase letters replaced by their corresponding lowercase letters.

2) When locale-name-1 is specified, the correspondence of uppercase to lowercase letters is determined from locale category LC_CTYPE in the locale associated with locale-name-1.

3) When locale-name-1 is not specified and a locale is in effect for character classification, as described in 12.3.5, OBJECT-COMPUTER paragraph, the correspondence of uppercase to lowercase letters is determined from locale category LC_CTYPE.

4) When a locale is not in effect, the implementor defines the correspondence of uppercase letters to lowercase letters.

5) The character string returned has the same length as argument-1 when there is a one-to-one correspondence between uppercase and lowercase letters. When the correspondence of uppercase and lowercase letters is not one-to-one, the character string returned may be longer or shorter than argument-1 and depends on the content of argument-1 and the correspondence rules that are in effect.

6) If there is no corresponding lowercase letter for a given uppercase letter, that letter is unchanged in the returned value; when a locale is in effect for character classification and there is no lowercase correspondence specified in the locale, the letter or letters are unchanged in the returned value.

15.54 LOWEST-ALGEBRAIC function

The LOWEST-ALGEBRAIC function returns a value that is equal to the lowest algebraic value that may be represented in argument-1.

The type of this function depends upon the argument types as follows:

Argument type	Function type
Alphanumeric	Numeric
Integer	Integer
National	Numeric
Numeric	Numeric

15.54.1 General format

FUNCTION LOWEST-ALGEBRAIC (argument-1)

15.54.2 Arguments

1) Argument-1 shall be a data item of category numeric or numeric-edited and shall not be an integer or numeric function.

15.54.3 Returned values

1) The value returned is equal to the lowest finite algebraic value that may be represented in argument-1.

NOTE The following illustrates the expected results for some values of argument-1.

Argument-1 characteristics	Value returned
S999	–999
S9(4) BINARY	–9999
99V9(3)	0
$**,**9.99BCR	–99999.99
$**,**9.99	0
BINARY-CHAR SIGNED	-128 (assuming an 8-bit twos-complement representation)
BINARY-CHAR UNSIGNED	0 (assuming an 8-bit twos-compliment representation)

15.55 MAX function

The MAX function returns the content of the argument-1 that contains the maximum value.

The type of this function depends upon the argument types as follows:

Argument type	**Function type**
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
Index	Index
All arguments integer	Integer
National	National
Numeric (some arguments may be integer)	Numeric

15.55.1 General format

FUNCTION MAX ({ argument-1 } ...)

15.55.2 Arguments

1) Argument-1 shall not be of class boolean, object, or pointer, nor shall it be a strongly-typed group item.

2) All arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

3) Argument-1 shall not be a zero-length literal.

15.55.3 Returned values

1) The returned value is the content of the argument-1 having the greatest value. The comparisons used to determine the greatest value are made according to the rules for simple conditions. (See 8.8.4.1, Simple conditions.)

2) If the value of more than one argument-1 is equal to the greatest value, the content of the argument-1 returned is the leftmost argument-1 having that value.

3) If the type of the function is alphanumeric or national, the size of the returned value is the same as the size of the selected argument-1.

15.56 MEAN function

The MEAN function returns a numeric value that is the arithmetic mean (average) of its arguments.

The type of this function is numeric.

15.56.1 General format

```
FUNCTION MEAN ( { argument-1 } ... )
```

15.56.2 Arguments

1) Argument-1 shall be of class numeric.

15.56.3 Returned values

1) The equivalent arithmetic expression is as follows:

   a) For one occurrence of argument-1,

      (argument-1)

   b) For two occurrences of argument-1,

      ((argument-1_1 + argument-1_2) / 2)

   c) For n occurrences of argument-1,

      ((argument-1_1 + argument-1_2 +... + argument-1_n) / n)

15.57 MEDIAN function

The MEDIAN function returns the content of the argument whose value is the middle value in the list formed by arranging the arguments in sorted order.

The type of this function is numeric.

15.57.1 General format

FUNCTION MEDIAN ({ argument-1 } ...)

15.57.2 Arguments

1) Argument-1 shall be of class numeric.

15.57.3 Returned values

1) When the number of occurrences of argument-1 is odd, the returned value is such that at least half of the occurrences referenced by argument-1 are greater than or equal to the returned value and at least half are less than or equal. For the purposes of the equivalent arithmetic expression, the middle value is referred to as argument-a. The equivalent arithmetic expression is

   (argument-a)

2) When the number of occurrences of argument-1 is even, the returned value is the arithmetic mean of the two middle values. For the purposes of the equivalent arithmetic expression, the two middle values are referred to as argument-b and argument-c. The equivalent arithmetic expression is

   ((argument-b + argument-c) / 2)

3) The comparisons used to arrange the argument-1 values in sorted order are made according to the rules for simple conditions. (See 8.8.4.1, Simple conditions.)

15.58 MIDRANGE function

The MIDRANGE (middle range) function returns a numeric value that is the arithmetic mean (average) of the values of the minimum argument and the maximum argument.

The type of this function is numeric.

15.58.1 General format

FUNCTION MIDRANGE ({ argument-1 } ...)

15.58.2 Arguments

1) Argument-1 shall be of class numeric.

15.58.3 Returned values

1) The equivalent arithmetic expression is

((FUNCTION MAX (argument-list) + FUNCTION MIN (argument-list)) / 2)

where argument-list is the argument-1 list for the MIDRANGE function itself.

15.59 MIN function

The MIN function returns the content of the argument-1 that contains the minimum value.

The type of this function depends upon the argument types as follows:

Argument Type	**Function type**
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
Index	Index
All arguments integer	Integer
National	National
Numeric (some arguments may be integer)	Numeric

15.59.1 General format

FUNCTION MIN ({ argument-1 } ...)

15.59.2 Arguments

1) Argument-1 shall not be of class boolean, object, or pointer, nor shall it be a strongly-typed group item.

2) All arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

3) Argument-1 shall not be a zero-length literal.

15.59.3 Returned values

1) The returned value is the content of the argument-1 having the least value. The comparisons used to determine the least value are made according to the rules for simple conditions. (See 8.8.4.1, Simple conditions.)

2) If the value of more than one argument-1 is equal to the least value, the content of the argument-1 returned is the leftmost argument-1 having that value.

3) If the type of the function is alphanumeric or national, the size of the returned value is the same as the size of the selected argument-1.

15.60 MOD function

The MOD function returns an integer value that is argument-1 modulo argument-2.

The type of this function is integer.

15.60.1 General format

FUNCTION MOD (argument-1 argument-2)

15.60.2 Arguments

1) Argument-1 and argument-2 shall be integers.

2) The value of argument-2 shall not be zero.

15.60.3 Returned values

1) The equivalent arithmetic expression is

((argument-1) – ((argument-2) * FUNCTION INTEGER ((argument-1) / (argument-2))))

where argument-1 and argument-2 for the INTEGER function are the same as the arguments for the MOD function itself.

NOTE The following illustrates the expected results for some values of argument-1 and argument-2.

Argument-1	Argument-2	Return
11	5	1
–11	5	4
11	–5	–4
–11	–5	–1

15.61 NATIONAL-OF function

The NATIONAL-OF function returns a character string containing the national character representation of the characters in the argument.

The type of the function is national.

15.61.1 General format

FUNCTION NATIONAL-OF (argument-1 [argument-2])

15.61.2 Arguments

1) Argument-1 shall be of class alphabetic or class alphanumeric.

2) Argument-2 shall be of category national and shall be one character in length. Argument-2 specifies a national substitution character for use in conversion of alphanumeric characters for which there is no corresponding national character.

3) Argument-1 shall not be a zero-length literal.

15.61.3 Returned values

1) A character string is returned with each alphanumeric character in argument-1 converted to its corresponding national coded character set representation. The implementor defines the correspondence of characters.

2) If argument-2 is specified, each character in argument-1 that has no corresponding national representation is converted to the substitution character specified by argument-2.

3) If argument-2 is unspecified and argument-1 contains an alphanumeric character for which there is no corresponding national character representation, an implementor-defined substitution character is used as the corresponding national character and the EC-DATA-CONVERSION exception condition is set to exist.

4) The length of the returned value is the number of character positions of usage national required to hold the converted argument and depends on the number of characters contained in argument-1.

15.62 NUMVAL function

The NUMVAL function returns the numeric value represented by the character string specified by argument-1. Leading and trailing spaces are ignored.

The type of this function is numeric.

NOTE Locale-based functionality equivalent to NUMVAL can be obtained by using the NUMVAL-C function with the LOCALE keyword. A currency sign is optional in NUMVAL-C. The locale category LC_MONETARY will be used because there is no sign convention specified in locale category LC_NUMERIC.

15.62.1 General format

FUNCTION NUMVAL (argument-1)

15.62.2 Arguments

1) Argument-1 shall be an alphanumeric or national literal or an alphanumeric or national data item whose content has one of the following two formats:

[space-string] [+ | –] [space-string] { digit [. [digit]] | . digit } [space-string]

or

[space-string] { digit [. [digit]] | . digit } [space-string] [+ | – | CR | DB] [space-string]

where space-string is a string of one or more space characters and digit is a string of one to 31 digits. If argument-1 is alphanumeric, CR or DB, if specified, shall be the letters 'CR' or 'DB' in uppercase or lowercase, or a combination of uppercase and lowercase, in the computer's alphanumeric character set. If argument-1 is national, CR or DB, if specified, shall be the letters 'CR' or 'DB' in uppercase or lowercase, or a combination of uppercase and lowercase, in the computer's national character set.

2) The total number of digits in argument-1 shall not exceed 31.

3) The character period in argument-1 represents the decimal separator. When the DECIMAL-POINT IS COMMA clause is specified, the character comma shall be used in argument-1 instead of the character period to represent the decimal separator

15.62.3 Returned values

1) The returned value is the numeric value represented by argument-1.

2) The maximum number of decimal digits represented in the returned value is 31.

3) If argument-1 contains CR, DB, or the minus sign, the returned value is negative.

15.63 NUMVAL-C function

The NUMVAL-C function returns the numeric value represented by the character string specified by argument-1. The currency string, if any, and any grouping separators preceding the decimal separator are ignored. Optionally, the currency string, the sign convention, the grouping separator, and the decimal separator permitted in the character string may be specified by locale category LC_MONETARY, or the currency string may be specified by argument-2.

The type of this function is numeric.

15.63.1 General format

```
FUNCTION NUMVAL-C ( argument-1 [ LOCALE [ locale-name-1 ] | argument-2 ] [ ANYCASE ] )
```

15.63.2 Arguments

1) Argument-1 shall be of category alphanumeric or national.

2) Argument-2, if specified, shall be of the same class as argument-1. Argument-2 shall contain at least one nonspace character. Any leading or trailing spaces in argument-2 are ignored. Argument-2 shall not contain any of the digits 0 through 9; the characters '*', '+', '–', ',' , or '.'; or the two consecutive letters 'CR' or 'DB', whether in uppercase or lowercase or a combination of uppercase and lowercase. Argument-2 specifies a currency string that may appear in argument-1.

   NOTE The currency string specified by argument-2 may contain spaces, in addition to the leading and trailing spaces.

3) If the ANYCASE keyword is specified, the matching rules for detecting a currency string in argument-1 are case-insensitive. If the ANYCASE keyword is not specified, the matching rules for detecting a currency string are case-sensitive.

4) If neither argument-2 nor the LOCALE keyword is specified, there shall be only one currency string for the compilation unit, either the default currency sign or a currency string specified in the SPECIAL-NAMES paragraph.

5) If the LOCALE keyword is not specified, the following rules apply:

   — Argument-1 shall have one of the following two formats:

```
[ space ] [ + | – ] [ space ] [ currency ] [ space ] { digit [ , digit ] ... [ . [ digit ] ] | . digit } [ space ]
```

   or

```
[ space ] [ currency ] [ space ] { digit [ , digit ] ... [ . [ digit ] ] | . digit } [ space ] [ + | – | CR | DB ] [ space ]
```

   where
   — digit is a string of one or more of the digits 0 through 9;
   — space is one or more space characters;

— currency is a string of one or more characters matching character for character the currency string in argument-2, if specified, or matching character for character the default currency sign if argument-2 is not specified.

a) If argument-1 is alphanumeric, CR or DB, if specified, shall be the letters 'CR' or 'DB' in uppercase or lowercase, or a combination of uppercase and lowercase, in the computer's alphanumeric character set.

b) If argument-1 is national, CR or DB, if specified, shall be the letters 'CR' or 'DB' in uppercase or lowercase, or a combination of uppercase and lowercase, in the computer's national character set.

c) The character period in argument-1 represents the decimal separator. The character comma in argument-1 represents the grouping separator. When the DECIMAL-POINT IS COMMA clause is specified, the character comma shall be used in argument-1 to represent the decimal separator and the character period shall be used to represent the grouping separator.

6) If the LOCALE keyword is specified, the following rules apply:

a) Locale-name-1, if specified, shall be associated with a locale in the SPECIAL-NAMES paragraph; locale category LC_MONETARY in the referenced locale is used in evaluating the monetary format in argument-1. If locale-name-1 is not specified, category LC_MONETARY in the current locale is used. If the required locale is not available, the EC-LOCALE-MISSING exception condition is set to exist.

b) The content of argument-1 shall be a string of digits and characters in a format consistent with the specifications of locale category LC_MONETARY in the locale in use. The following rules apply:

1. If the ANYCASE keyword is specified, the matching rules for detecting a currency string in argument-1 are case-insensitive. If the ANYCASE keyword is not specified, the matching rules for detecting a currency string are case-sensitive.

2. Usage national representation of locale fields is used for purposes of matching argument-1. If argument-1 is of category alphanumeric, usage national representation of argument-1 is used for purposes of matching locale fields.

3. Argument-1 may contain a currency string that matches either locale field currency_symbol or the first three characters of locale field int_curr_symbol, in accordance with locale fields p_cs_precedes and n_cs_precedes.

4. Argument-1 may contain a positive sign in accordance with locale fields positive_sign and p_sign_posn or a negative sign in accordance with locale fields negative_sign and n_sign_posn.

5. Argument-1 may contain a decimal separator in accordance with locale fields mon_decimal_point, int_frac_digits, and frac_digits.

6. Argument-1 may contain one or more grouping separators in accordance with locale fields mon_thousands_sep and mon_grouping.

7. Argument-1 shall contain at least one digit and may contain leading or trailing spaces.

7) The total number of digits in argument-1 shall not exceed 31.

15.63.3 Returned values

1) The returned value is 31 digits and is the numeric value represented by argument-1.

2) When the LOCALE keyword is specified, the returned value is negative if argument-1 contains a negative sign convention as specified by locale fields negative_sign and n_sign_posn. When the LOCALE keyword is not specified, the returned value is negative if argument-1 contains CR, DB, or a minus sign.

15.64 NUMVAL-F function

The NUMVAL-F function returns the value or an approximation of the value represented by the character string specified by argument-1. Leading, trailing, and embedded spaces are ignored.

The type of this function is numeric.

15.64.1 General format

FUNCTION NUMVAL-F (argument-1)

15.64.2 Arguments

1) Argument-1 shall be an alphanumeric or national literal or data item whose content has the following format:

$$[\ \text{space}\ ]\ \left[\begin{array}{c}+\\-\end{array}\right]\ [\ \text{space}\ ]\ \left\{\begin{array}{l}\text{digit}\ [\ .\ [\ \text{digit}\ ]\ ]\\ .\ \text{digit}\end{array}\right\}\ [\ \text{space}\ ]\ \left[\ \text{E}\ [\ \text{space}\ ]\ \left\{\begin{array}{c}+\\-\end{array}\right\}\ [\ \text{space}\ ]\ \text{n}\ [\ \text{space}\ ]\ \right]$$

where space is one or more space characters; n is one, two, three, or four digits representing the exponent; and digit is a string of 1 to 34 digits. If argument-1 is alphanumeric, E shall be either an uppercase or lowercase E in the computer's alphanumeric character set. If argument-1 is national, E shall be either an uppercase or lowercase E in the computer's national character set.

2) If standard arithmetic or native arithmetic is in effect, the total number of digits in the significand shall not exceed 31.

3) If standard-binary or standard-decimal arithmetic is in effect, the total number of digits in the significand shall not exceed 34.

4) The character period in argument-1 represents the decimal separator. When the DECIMAL-POINT IS COMMA clause is specified, the character comma shall be used in argument-1 instead of the character period to represent the decimal separator.

15.64.3 Returned values

1) Leading, trailing, and embedded spaces are ignored.

2) If native arithmetic is in effect, the returned value is an approximation of the numeric value represented by argument-1. If standard arithmetic, standard-binary arithmetic, or standard-decimal arithmetic is in effect, the returned value is the numeric value represented by argument-1.

15.65 ORD function

The ORD function returns an integer value that is the ordinal position of argument-1 in the program collating sequence. The lowest ordinal position is 1.

The type of this function is integer.

15.65.1 General format

FUNCTION ORD (argument-1)

15.65.2 Arguments

1) Argument-1 shall be one character position in length and shall be of category alphabetic, alphanumeric, or national.

15.65.3 Returned values

1) If the class of argument-1 is alphabetic or alphanumeric, the returned value is the ordinal position of argument-1 in the current alphanumeric program collating sequence.

2) If the class of argument-1 is national, the returned value is the ordinal position of argument-1 in the current national program collating sequence.

15.66 ORD-MAX function

The ORD-MAX function returns a value that is the ordinal number of the argument-1 that contains the maximum value.

The type of this function is integer.

15.66.1 General format

FUNCTION ORD-MAX ({ argument-1 } ...)

15.66.2 Arguments

1) Argument-1 shall not be of class boolean, object, or pointer, nor shall it be a strongly-typed group item.

2) Argument-1 shall not be a zero-length literal.

3) All arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

15.66.3 Returned values

1) The returned value is the ordinal number that corresponds to the position of the argument-1 having the greatest value in the argument-1 series.

2) The comparisons used to determine the greatest valued argument are made according to the rules for simple conditions. (See 8.8.4.1, Simple conditions.)

3) If the value of more than one argument-1 is equal to the greatest value, the number returned corresponds to the position of the leftmost argument-1 having that value.

15.67 ORD-MIN function

The ORD-MIN function returns a value that is the ordinal number of the argument that contains the minimum value.

The type of this function is integer.

15.67.1 General format

FUNCTION ORD-MIN ({ argument-1 } ...)

15.67.2 Arguments

1) Argument-1 shall not be of class boolean, object, or pointer, nor shall it be a strongly-typed group item.

2) Argument-1 shall not be a zero-length literal.

3) All arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

15.67.3 Returned values

1) The returned value is the ordinal number that corresponds to the position of the argument-1 having the least value in the argument-1 series.

2) The comparisons used to determine the least valued argument-1 are made according to the rules for simple conditions. (See 8.8.4.1, Simple conditions.)

3) If the value of more than one argument-1 is equal to the least value, the number returned corresponds to the position of the leftmost argument-1 having that value.

15.68 PI function

The PI function returns a value that is an approximation of π, the ratio of the circumference of a circle to its diameter.

The type of this function is numeric.

15.68.1 General format

FUNCTION PI

15.68.2 Returned values

1) If native arithmetic is in effect, the returned value is an implementor-defined approximation of the arithmetic expression

   (3 + 0.14159265358979323846264338327950)

2) If standard arithmetic is in effect, the equivalent arithmetic expression is

   (3 + 0.14159265358979323846264338327950)

3) If standard-binary arithmetic is in effect, the returned value is the exact value of the arithmetic expression

   (16,312,081,666,030,376,401,667,486,162,748,272 / (2 ** 112))

   in Binary128 format, normalized as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings.

4) If standard-decimal arithmetic is in effect, the returned value is exactly

   3.14159265358979323846264338327950 3

   in decimal128 format with decimal encoding of the significand, as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.5, Decimal interchange format encodings.

15.69 PRESENT-VALUE function

The PRESENT-VALUE function returns a value that approximates the present value of a series of future period-end amounts specified by argument-2 at a discount rate specified by argument-1.

The type of this function is numeric.

15.69.1 General format

FUNCTION PRESENT-VALUE (argument-1 { argument-2 } ...)

15.69.2 Arguments

1) Argument-1 and argument-2 shall be of the class numeric.

2) The value of argument-1 shall be greater than –1.

15.69.3 Returned values

1) The equivalent arithmetic expression is as follows:

   a) For one occurrence of argument-2,

      (argument-2 / (1 + argument-1))

   b) For two occurrences of argument-2,

      (argument-2_1 / (1 + argument-1) + argument-2_2 / (1 + argument-1) ** 2)

   c) For n occurrences of argument-2, the equivalent arithmetic expression is

      (FUNCTION SUM (

      (argument-2_1 / (1 + argument-1) ** 1)

      ...

      (argument-2_n / (1 + argument-1) ** n)))

   where argument-1 and argument-2_i in the terms of the SUM function are the same as the arguments for the PRESENT-VALUE function itself.

15.70 RANDOM function

The RANDOM function returns a numeric value that is a pseudo-random number from a rectangular distribution.

The type of this function is numeric.

15.70.1 General format

```
FUNCTION RANDOM [ ( [ argument-1 ] ) ]
```

15.70.2 Arguments

1) Argument-1 shall be of class numeric.

2) If argument-1 is specified, it shall be zero or a positive integer. It is used as the seed value to generate a sequence of pseudo-random numbers.

3) If a subsequent reference specifies argument-1, a new sequence of pseudo-random numbers is started.

4) If the first reference to this function in the run unit does not specify argument-1, the seed value is defined by the implementor.

5) In each case, subsequent references without specifying argument-1 return the next number in the current sequence.

15.70.3 Returned values

1) The returned value is greater than or equal to zero and less than one.

2) For a given seed value on a given implementation, the sequence of pseudo-random numbers will always be the same.

3) The implementor shall specify the subset of the domain of argument-1 values that will yield distinct sequences of pseudo-random numbers. This subset shall include the values from 0 through at least 32767.

15.71 RANGE function

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument.

The type of this function depends upon the argument types as follows:

Argument type	Function type
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

15.71.1 General format

FUNCTION RANGE ({ argument-1 } ...)

15.71.2 Arguments

1) Argument-1 shall be of class numeric.

15.71.3 Returned values

1) The equivalent arithmetic expression is

(FUNCTION MAX (argument-list) – FUNCTION MIN (argument-list))

where argument-list is the argument-1 list for the RANGE function itself.

15.72 REM function

The REM function returns a numeric value that is the remainder of argument-1 divided by argument-2.

The type of this function is numeric.

15.72.1 General format

```
FUNCTION REM ( argument-1  argument-2 )
```

15.72.2 Arguments

1) Argument-1 and argument-2 shall be of class numeric.

2) The value of argument-2 shall not be zero.

15.72.3 Returned values

1) The equivalent arithmetic expression is

    ((argument-1) – ((argument-2) * FUNCTION INTEGER-PART ((argument-1) / (argument-2))))

    where argument-1 and argument-2 of the INTEGER-PART function are the same as the arguments for the REM function itself.

15.73 REVERSE function

The REVERSE function returns a character string of exactly the same length as argument-1 and whose characters are exactly the same as those of argument-1, except that they are in reverse order.

The type of the function depends on the argument type as follows:

Argument type	**Function type**
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
National	National

15.73.1 General format

FUNCTION REVERSE (argument-1)

15.73.2 Arguments

1) Argument-1 shall be of class alphabetic, alphanumeric, or national and shall be at least one character position in length.

15.73.3 Returned values

1) If argument-1 is a character string of length n, the returned value is a character string of length n such that for $1 \leq j \leq n$, the character in position j of the returned value is the character from position $n - j + 1$ of argument-1.

15.74 SECONDS-FROM-FORMATTED-TIME function

The SECONDS-FROM-FORMATTED-TIME function converts a time that is in a specified format to a numeric value representing the number of seconds after midnight.

The type of the function is numeric.

15.74.1 General Format

FUNCTION SECONDS-FROM-FORMATTED-TIME (argument-1 argument-2)

15.74.2 Arguments

1) Argument-1 shall be a national or alphanumeric literal.

2) The content of argument-1 shall be either a time format or a combined date and time format.

3) Argument-2 shall have the same type as argument-1.

4) If argument-1 is a time format, the contents of argument-2 shall be a time in that format.

5) If argument-1 is a combined date and time format, the contents of argument-2 shall be a valid date and time in that format.

15.74.3 Returned Values

1) The equivalent arithmetic expression is as follows:

    ((H * 3600) + (M * 60) + S)

    where H is the portion of argument-2 corresponding to the hours subfield of the format in argument-1, M is the portion of argument-2 corresponding to the minutes subfield of argument-1, and S is the portion of argument-2 corresponding to the seconds subfield of argument-1.

15.75 SECONDS-PAST-MIDNIGHT function

The SECONDS-PAST-MIDNIGHT function returns a value in standard numeric time form that represents the current local time of day provided by the system on which the function is evaluated.

The type of this function is numeric.

15.75.1 General format

FUNCTION SECONDS-PAST-MIDNIGHT

15.75.2 Returned values

1) The returned value is in standard numeric time form.

2) The returned value is the current local time of day provided by the system on which the function is evaluated, expressed in seconds past midnight.

3) The implementor shall specify the precision to which this value is returned.

4) The implementor shall specify whether the returned value may be greater than or equal to 86,400 when the LEAP-SECOND directive with the ON phrase is in effect.

15.76 SIGN function

The SIGN function returns +1, 0, or –1 depending on the sign of the argument.

The type of the function is integer.

15.76.1 General Format

FUNCTION SIGN (argument-1)

15.76.2 Arguments

1) Argument-1 shall be of class numeric.

15.76.3 Returned Values

1) The equivalent arithmetic expression is as follows:

   a) When the value of argument-1 is positive,

      (1)

   b) When the value of argument-1 is zero,

      (0)

   c) When the value of argument-1 is negative,

      (–1)

15.77 SIN function

The SIN function returns a numeric value that approximates the sine of an angle or arc, expressed in radians, that is specified by argument-1.

The type of this function is numeric.

15.77.1 General format

FUNCTION SIN (argument-1)

15.77.2 Arguments

1) Argument-1 shall be of class numeric.

15.77.3 Returned values

1) The returned value is the approximation of the sine of argument-1 and is greater than or equal to –1 and less than or equal to +1.

15.78 SQRT function

The SQRT function returns a numeric value that approximates the square root of argument-1.

The type of this function is numeric.

15.78.1 General format

FUNCTION SQRT (argument-1)

15.78.2 Arguments

1) Argument-1 shall be of class numeric.

2) The value of argument-1 shall be zero or positive.

15.78.3 Returned values

1) When standard arithmetic, standard-binary arithmetic, or standard-decimal arithmetic is in effect, argument-1 is not rounded.

2) When standard arithmetic is in effect, the returned value is the absolute value of the exact square root of argument-1 truncated to 32 digits, normalized, and stored in a standard intermediate data item.

   NOTE When standard arithmetic is in effect, there is no equivalent arithmetic expression. The arithmetic expression operand-1 ** 0.5 is evaluated according to the rules for the evaluation of arithmetic expressions.

3) When standard-decimal arithmetic is in effect, the returned value is the absolute value of the exact square root of argument-1 rounded to 34 digits according to the rules for standard-decimal arithmetic and stored in a standard-decimal intermediate data item.

4) When standard-binary arithmetic is in effect, the returned value is the absolute value of the exact square root of argument-1 rounded to 113 significant bits in the significand according to the rules for standard-binary arithmetic and stored in a standard-binary intermediate data item.

5) When native arithmetic is in effect, the returned value is the absolute value of the approximation of the square root of argument-1.

15.79 STANDARD-COMPARE function

The STANDARD-COMPARE function returns a character indicating the result of comparing argument-1 and argument-2 using a culturally-sensitive ordering table constructed in compliance with ISO/IEC 14651.

The function type is alphanumeric.

15.79.1 General format

FUNCTION STANDARD-COMPARE (argument-1 argument-2 [ordering-name-1] [argument-4])

15.79.2 Arguments

1) Argument-1 shall be of class alphabetic, alphanumeric, or national.

2) Argument-2 shall be of class alphabetic, alphanumeric, or national.

3) Argument-1 and argument-2 may be of different classes.

4) Neither argument-1 nor argument-3 shall be a zero-lenght literal.

5) Ordering-name-1, if specified, shall be associated with an ordering table in the ORDER TABLE clause of the SPECIAL-NAMES paragraph. Ordering-name-1 identifies the ordering table to be used for the comparison. If ordering-name-1 is not specified, the default ordering table 'ISO14651_2006_TABLE1' specified in ISO/IEC 14651 shall be used.

6) Argument-4, if specified, shall be a positive nonzero integer.

15.79.3 Returned values

1) If argument-4 is unspecified, the highest level defined in the ordering table is used for the comparison.

2) If ISO/IEC 14651 is not available on the processor, or the specified ordering level is not supported, or the level number specified by argument-4 is not defined in the ordering table, the EC-ORDER-NOT-SUPPORTED exception condition is set to exist.

3) If the arguments are of different classes, and one is national, the other argument is converted to class national for purposes of comparison.

4) For purposes of comparison, trailing spaces are truncated from the operands except that an operand consisting of all spaces is truncated to a single space.

5) Argument-1 and argument-2 are compared in accordance with the ordering table and ordering level being used, as specified in ISO/IEC 14651.

   NOTE This comparison is culturally sensitive and acceptable for most cultures. It is not necessarily a character-by-character comparison and not necessarily a case-sensitive comparison. In order to use this function, users should understand the types of comparisons specified by ISO/IEC 14651 and the ordering tables in use for their installation.

6) The returned value is:
   "=" if the arguments compare equal,
   "<" if argument-1 is less than argument-2,
   ">" if argument-1 is greater than argument-2.

7) The length of the returned value is 1.

15.80 STANDARD-DEVIATION function

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments.

The type of this function is numeric.

15.80.1 General format

FUNCTION STANDARD-DEVIATION ({ argument-1 } ...)

15.80.2 Arguments

1) Argument-1 shall be of class numeric.

15.80.3 Returned values

1) The equivalent arithmetic expression is as follows:

(FUNCTION SQRT (FUNCTION VARIANCE (argument-list)))

where argument-list is the argument-1 list for the STANDARD-DEVIATION function itself.

15.81 SUM function

The SUM function returns a value that is the sum of the arguments.

The type of this function depends upon the argument types as follows:

Argument type	Function type
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

15.81.1 General format

FUNCTION SUM ({ argument-1 } ...)

15.81.2 Arguments

1) Argument-1 shall be of class numeric.

15.81.3 Returned values

1) The equivalent arithmetic expression is as follows:

   a) For one occurrence of argument-1,

      (argument-1)

   b) For two occurrences of argument-1,

      (argument-1_1 + argument-1_2)

   c) For n occurrences of argument-1,

      (argument-1_1 + argument-1_2 + ... + argument-1_n)

15.82 TAN function

The TAN function returns a numeric value that approximates the tangent of an angle or arc, expressed in radians, that is specified by argument-1.

The type of this function is numeric.

15.82.1 General format

FUNCTION TAN (argument-1)

15.82.2 Arguments

1) Argument-1 shall be of class numeric.

15.82.3 Returned values

1) The returned value is the approximation of the tangent of argument-1.

15.83 TEST-DATE-YYYYMMDD function

The TEST-DATE-YYYYMMDD function tests whether a date in standard date form (YYYYMMDD) is a valid date in the Gregorian calendar. Argument-1 of the INTEGER-OF-DATE function is required to be in standard date form.

The type of this function is integer.

15.83.1 General Format

FUNCTION TEST-DATE-YYYYMMDD (argument-1)

15.83.2 Arguments

1) Argument-1 shall be an integer.

15.83.3 Returned values

1) The returned value is:

   a) If the value of argument-1 is less than 16010000 or greater than 99999999,

      (1)

      NOTE 1 The year is not within the range 1601 to 9999.

   b) Otherwise, if the value of FUNCTION MOD (argument-1 10000) is less than 100 or greater than 1299,

      (2)

      NOTE 2 The month is not within the range 1 through 12.

   c) Otherwise, if the value of FUNCTION MOD (argument-1 100) is less than 1 or greater than the number of days in the month determined by FUNCTION INTEGER (FUNCTION MOD (argument-1 10000) / 100) of the year determined by FUNCTION INTEGER (argument-1 / 10000),

      (3)

      NOTE 3 The day is not valid for the given year and month.

   d) Otherwise,

      (0)

      NOTE 4 The date is valid.

15.84 TEST-DAY-YYYYDDD function

The TEST-DAY-YYYYDDD function tests whether a date in Julian date form (YYYYDDD) is a valid date in the Gregorian calendar. Argument-1 of the INTEGER-OF-DAY function is required to be in Julian date form.

The type of this function is integer.

15.84.1 General Format

FUNCTION TEST-DAY-YYYYDDD (argument-1)

15.84.2 Arguments

1) Argument-1 shall be an integer.

15.84.3 Returned values

1) The returned value is:

   a) If the value of argument-1 is less than 1601000 or greater than 9999999,

      (1)

      NOTE 1 The year is not within the range 1601 to 9999.

   b) Otherwise, if the value of FUNCTION MOD (argument-1 1000) is less than 1 or greater than the number of days in the year determined by FUNCTION INTEGER (argument-1 / 1000),

      (2)

      NOTE 2 The day is not valid in the given year.

   c) Otherwise,

      (0)

      NOTE 3 The date is valid.

15.85 TEST-FORMATTED-DATETIME function

The TEST-FORMATTED-DATETIME function tests whether a data item representing a date, a time, or a combined date and time is valid according to the specified format.

The type of this function is integer.

15.85.1 General Format

FUNCTION TEST-FORMATTED-DATETIME (argument-1 argument-2)

15.85.2 Arguments

1) Argument-1 shall be a national or alphanumeric literal.

2) The content of argument-1 shall be either a date format, a time format, or a combined date and time format.

3) Argument-2 shall be of the same type as argument-1.

15.85.3 Returned values

1) If no format problems or range problems occur during the evaluation of argument-2 according to the format in argument-1, the value returned is zero. Otherwise, the value returned is the ordinal character position at which the first error in argument-2 was detected.

NOTE For example, given the declaration

```
A-DATE PICTURE X(8) VALUE "20031314".
```

the value returned from

```
FUNCTION TEST-FORMATTED-DATETIME ("YYYYMMDD", A-DATE)
```

is

(6).

If a statement of the form MOVE "15990315" TO A-DATE is then executed, the value returned from the next function reference of the same form is

(2).

15.86 TEST-NUMVAL function

The TEST-NUMVAL function verifies that the contents of argument-1 conform to the specification for argument-1 of the NUMVAL function.

The type of this function is integer.

15.86.1 General Format

FUNCTION TEST-NUMVAL (argument-1)

15.86.2 Arguments

1) Argument-1 shall be an alphanumeric or national literal or a data item of class alphanumeric or national.

15.86.3 Returned values

1) The returned value is:

   a) If the content of argument-1 conforms to the argument rules for the NUMVAL function,

      (0)

   b) Otherwise, if one or more characters are in error, the position of the first character in error,

      NOTE 1 Because one or more spaces following one or more digits is valid, if one or more spaces are embedded within a string of numeric characters, the returned value is the position of the first nonspace character following the spaces. If argument-1 is '0 1', the returned value will be 3.

      NOTE 2 If either standard arithmetic or native arithmetic is in effect, because the character in error for an argument that is greater that 31 digits is the 32nd digit, the returned value is the position of the 32nd digit if no prior error is found.

      NOTE 3 If either standard-binary or standard-decimal arithmetic is in effect, because the character in error for an argument that is greater that 34 digits, the returned value is the position of the 35th digit if no prior error is found.

   c) Otherwise,

      (FUNCTION LENGTH (argument-1) + 1).

      NOTE 4 These errors include, but are not limited to:

      – argument-1 is zero-length,

      – argument-1 contains only spaces,

      – argument-1 contains valid characters but is incomplete, such as the string ' +.'.

15.87 TEST-NUMVAL-C function

The TEST-NUMVAL-C function verifies that the contents of argument-1 conform to the specification for argument-1 of the NUMVAL-C function. The purpose of this function is to allow the user verify that the NUMVAL-C function will produce a valid numeric result for a given set of arguments.

The type of this function is integer.

15.87.1 General Format

FUNCTION TEST-NUMVAL-C (argument-1 [LOCALE [locale-name-1] | argument-2] [ANYCASE])

15.87.2 Arguments

1) The argument rules for the TEST-NUMVAL-C function are the same as those specified in 15.63, NUMVAL-C function, Arguments.

15.87.3 Returned values

1) The returned value is:

   a) If the content of argument-1 conforms to the argument rules for the NUMVAL-C function,

      (0)

   b) Otherwise, if one or more characters are in error, the position of the first character in error,

      NOTE 1 Because one or more spaces following one or more digits is valid, if one or more spaces are embedded within a string of numeric characters, the returned value is the position of the first nonspace character following the spaces. If argument-1 is '0 1', the returned value will be 3.

      NOTE 2 Because the character in error for an argument that is greater that 31 digits is the 32nd digit, the returned value is the position of the 32nd digit if no prior error is found.

   c) Otherwise,

      (FUNCTION LENGTH (argument-1) + 1).

      NOTE 3 These errors include, but are not limited to:

      – argument-1 is zero-length,

      – argument-1 contains only spaces,

      – argument-1 contains valid characters but is incomplete, such as the string ' +.'.

15.88 TEST-NUMVAL-F function

The TEST-NUMVAL-F function verifies that the contents of argument-1 conform to the specification for argument-1 of the NUMVAL-F function.

The type of this function is integer.

15.88.1 General Format

FUNCTION TEST-NUMVAL-F (argument-1)

15.88.2 Arguments

1) Argument-1 shall be an alphanumeric or national literal or a data item of class alphanumeric or national.

15.88.3 Returned values

1) The returned value is:

   a) If the content of argument-1 conforms to the argument rules for the NUMVAL-F function,

      (0)

   b) Otherwise, if one or more characters are in error, the position of the first character in error,

      NOTE 1 Because one or more spaces following one or more digits is valid, if one or more spaces are embedded within a string of numeric characters, the returned value is the position of the first nonspace character following the spaces. If argument-1 is '0 1E+2', the returned value will be 3.

      NOTE 2 In the event that either standard arithmetic or native arithmetic is in effect, the argument contains a significand longer than 31 digits, and no prior error in the argument has been found, the returned value is the position of the 32nd digit of the significand because the character in error for the significand of an argument that is greater than 31 digits is the 32nd digit. For the same reason, the character in error for the exponent of an argument that is greater than 3 digits is the position of the 4th digit of the exponent.

      NOTE 3 In the event that either standard-binary or standard-decimal arithmetic is in effect, the argument contains a significand longer than 34 digits, and no prior error in the argument has been found, the returned value is the position of the 35th digit of the significand because the character in error for the significand of an argument that is greater than 34 digits is the 35th digit. For the same reason, the character in error for the exponent of an argument that is greater than 3 digits is the position of the 4th digit of the exponent.

      NOTE 4 If there is no sign in the exponent of an argument, the position of the first digit of the exponent would be the character in error.

   c) Otherwise,

      (FUNCTION LENGTH (argument-1) + 1).

      NOTE 5 These errors include, but are not limited to:

      – argument-1 is zero-length,

      – argument-1 contains only spaces,

      – argument-1 contains valid characters but is incomplete, such as the string ' +.'.

15.89 TRIM function

The TRIM function returns a character string that contains the characters in the argument with leading spaces, trailing spaces, or both, deleted.

The type of the function depends on the argument type as follows:

Argument type	Function type
Alphanumeric	Alphanumeric
National	National

15.89.1 General format

FUNCTION TRIM (argument-1 [LEADING / TRAILING])

15.89.2 Arguments

1) Argument-1 shall be a data item of class alphanumeric or national.

15.89.3 Returned values

1) If LEADING is specified, the returned value is a character string that consists of the characters in argument-1 beginning from the leftmost character position that does not contain a space character through the rightmost character position.

2) If TRAILING is specified, the returned value is a character string that consists of the characters in argument-1 beginning from the leftmost character position through the rightmost character position that does not contain a space character.

3) If neither LEADING nor TRAILING is specified, the returned value is a character string that consists of the characters in argument-1 beginning from the leftmost character position that does not contain a space character through the rightmost character position that does not contain a space character.

4) If argument-1 contains all spaces or argument-1 is of length zero, the returned value is of length zero.

15.90 UPPER-CASE function

The UPPER-CASE function returns a character string that contains the value of argument-1 with any lowercase letters in the argument replaced by their corresponding uppercase letters.

The type of the function depends on the argument type as follows:

Argument type	Function type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
National	National

15.90.1 General format

FUNCTION UPPER-CASE (argument-1 [LOCALE locale-name-1])

15.90.2 Arguments

1) Argument-1 shall be of class alphabetic, alphanumeric, or national and shall be at least one character position in length.

15.90.3 Returned values

1) A character string with the content of argument-1 is returned, with any lowercase letters replaced by their corresponding uppercase letters.

2) When locale-name-1 is specified, the correspondence of lowercase to uppercase letters is determined from locale category LC_CTYPE in the locale associated with locale-name-1.

3) When a locale is in effect for character classification, as described in 12.3.5, OBJECT-COMPUTER paragraph, and locale-name-1 is not specified, the correspondence of lowercase to uppercase letters is determined from locale category LC_CTYPE.

4) When a locale is not in effect, the implementor defines the correspondence of lowercase letters to uppercase letters.

5) The character string returned has the same length as argument-1 when there is a one-to-one correspondence between lowercase and uppercase letters. When the correspondence of lowercase and uppercase letters is not one-to-one, the character string returned may be longer or shorter than argument-1 and depends on the content of argument-1 and the correspondence rules that are in effect.

6) If there is no corresponding uppercase letter for a given lowercase letter, that letter is unchanged in the returned value; when a locale is in effect for character classification and there is no uppercase correspondence specified in the locale for a given letter or letters, the letter or letters are unchanged in the returned value.

15.91 VARIANCE function

The VARIANCE function returns a numeric value that approximates the variance of its arguments.

The type of this function is numeric.

15.91.1 General format

FUNCTION VARIANCE ({ argument-1 } ...)

15.91.2 Arguments

1) Argument-1 shall be of class numeric.

15.91.3 Returned values

1) The equivalent arithmetic expression is as follows:

   a) For one occurrence of argument-1,

      (0)

   b) For two occurrences of argument-1,

      (((argument-1_1 – FUNCTION MEAN (argument-list)) ** 2 +
      (argument-1_2– FUNCTION MEAN (argument-list)) ** 2) / 2)

   c) For n occurrences of argument-1,

      (FUNCTION SUM (

      ((argument-1_1 – FUNCTION MEAN (argument-list)) ** 2)

      ...

      ((argument-1_n – FUNCTION MEAN (argument-list)) ** 2)) / n)

   where argument-list is the argument-1 list for the VARIANCE function itself and argument-1_i is the ith argument of the argument-1 list for the VARIANCE function itself.

15.92 WHEN-COMPILED function

The WHEN-COMPILED function returns the date and time the compilation unit was compiled as provided by the system on which the compilation unit was compiled.

The type of this function is alphanumeric.

15.92.1 General format

FUNCTION WHEN-COMPILED

15.92.2 Returned values

1) The character positions returned, numbered from left to right, are:

Character Positions	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric characters of the seconds past the minute in the range: - 00 through 59 when a LEAP-SECOND directive with the OFF phrase is in effect - 00 through nn, where nn is defined by the implementor, when a LEAP-SECOND directive with the ON phrase is in effect.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the compilation was done does not have the facility to provide the fractional part of a second.
17	Either the character '–', the character '+', or the character '0'. The character '–' is returned if the local time indicated in the previous character positions is behind Coordinated Universal Time. The character '+' is returned if the local time indicated is the same as or ahead of Coordinated Universal Time. The character '0' is returned if the system on which the compilation was done does not have the facility to provide the local time differential factor.
18-19	If character position 17 is '–', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the local time is behind Coordinated Universal Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the local time is ahead of Coordinated Universal Time. If character position 17 is '0', the value 00 is returned.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the local time is ahead of or behind Coordinated Universal Time, depending on whether character position 17 is '+' or '–', respectively. If character position 17 is '0', the value 00 is returned.

2) The returned value is the date and time of compilation of the compilation unit that contains this function. The returned value in a contained source unit is the compilation date and time associated with the compilation unit in which it is contained.

3) The returned value shall denote the same time as the compilation date and time if provided in the listing and in the generated object code, although their representations and precision may differ.

15.93 YEAR-TO-YYYY function

The YEAR-TO-YYYY function converts argument-1, the two low-order digits of a year, to a four-digit year. Argument-2, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding window, into which the year of argument-1 falls. Argument-3 specifies the year at the time of execution.

The type of the function is integer.

15.93.1 General format

FUNCTION YEAR-TO-YYYY (argument-1 [argument-2 [argument-3]])

15.93.2 Arguments

1) Argument-1 shall be a nonnegative integer that is less than 100.

2) Argument-2 shall be an integer.

3) If argument-2 is omitted, the function shall be evaluated as though 50 were specified for argument-2.

4) Argument-3 shall be an integer greater than 1600 and less than 10000.

5) If argument-3 is omitted, the function shall be evaluated as though the following were specified for argument-3:

   FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4)))

6) The sum of the values of argument-2 and argument-3 shall be less than 10000 and greater than 1699.

15.93.3 Returned values

1) Maximum-year is calculated as follows:

   (argument-2 + argument-3)

2) The equivalent arithmetic expression is as follows:

   a) When the following condition is true

      FUNCTION MOD (maximum-year, 100) >= argument-1

      The equivalent arithmetic expression is

      (argument-1 + 100 * (FUNCTION INTEGER (maximum-year/100)))

   b) Otherwise, the equivalent arithmetic expression is

      (argument-1 + 100 * (FUNCTION INTEGER (maximum-year/100) – 1))

      NOTE 1 In the year 1995, the returned value for FUNCTION YEAR-TO-YYYY (4, 23) is 2004. In the year 2008 the returned value for FUNCTION YEAR-TO-YYYY (98, (–15)) is 1898.

      NOTE 2 If argument-3 is omitted, the YEAR-TO-YYYY function implements a sliding window algorithm, based on the year at the time of execution, as returned by the CURRENT-DATE function. A fixed window algorithm can be achieved by specifying suitable values for argument-2 and argument-3, such that the sum of argument-2 and argument-3 defines the ending year of the desired 100-year interval.

16 Standard classes

16.1 General

A standard class BASE shall be provided by the implementation. It may be used as the root of a class hierarchy to provide standard object life-cycle function. This use is not required — an implementation may provide alternative root classes that use other mechanisms for supporting the object life-cycle, particularly for creating COBOL objects that interoperate with other non-COBOL object systems.

16.2 BASE class

The following is the specification of the formal interfaces that are supported by the standard BASE class. The interface BaseFactoryInterface specifies the factory interface of the BASE class, and the interface BaseInterface specifies the object interface of the BASE class. The implementation of the BASE class shall be described with an IMPLEMENTS clause that references the interface defined here, and shall provide the semantics specified below.

NOTE The standard class BASE need not be implemented in COBOL.

```
Interface-id. BaseFactoryInterface.
Procedure division.
   Method-id. New.
   Data division.
   Linkage section.
   01 outObject usage object reference active-class.
   Procedure division returning outObject.
   End method New.
End Interface BaseFactoryInterface.

Interface-id. BaseInterface.
Procedure division.
   Method-id. FactoryObject.
   Data division.
   Linkage section.
   01 outFactory usage object reference factory of active-class.
   Procedure division returning outFactory.
   End method FactoryObject.
End Interface BaseInterface.
```

16.2.1 New method

The New method is a factory method that provides a standard mechanism for creating instance objects of a class.

16.2.1.1 General rules

1) The New method allocates storage for an object, initializes its instance data in accordance with 14.6.2.2, Initial state of object data, and returns a reference to the created object.

2) If resources needed to create a new object are not available, the returned object reference is set to NULL, and the EC-OO-RESOURCE exception condition is set to exist and is propagated back to the runtime element that invoked the New method.

16.2.2 FactoryObject method

The FactoryObject method is an instance method that provides a standard mechanism for acquiring access to the factory object associated with the class of a given instance.

16.2.2.1 General rules

1) When invoked on an instance object, the FactoryObject method determines the class of the object and returns a reference to the factory object associated with that class.

NOTE This method is useful when one does not know the class of an object. If the class is known, one can access the methods of a factory object trivially, using statements such as:

```
Invoke classname "someFactoryMethodName".
```

If the class of the object referenced by identifier anObject is unknown, then the coding below may be used to invoke one of its factory methods:

```
Invoke anObject "FactoryObject" returning aFactoryObject
Invoke aFactoryObject "someFactoryMethodName".
```

Annex A
(normative)

Language element lists

A.1 Implementor-defined language element list

The following is a list of the language elements within this final committee draft International Standard that depend on implementor definition to complete the specification of the elements. Each element is defined as required, optional, or conditionally required. Furthermore, each element is defined as requiring (or not requiring) user documentation. These terms have the following meaning:

— Required: The element shall be provided by the implementor. When the element is part of a feature that is optional or processor-dependent, the item is not required if the optional or processor-dependent feature is not implemented.

— Optional: The element may be provided at the implementor's option.

— Conditionally required: If the associated feature or language element is implemented then this element is also required.

— Documentation required: If the element is provided by the implementor, the implementor's user documentation shall document the element or shall reference other documentation that fulfills this requirement.

A short header and informative optional parenthetical text provide a paraphrase of the normative detailed specification located in the body of this final committee draft International Standard and direct the reader to that detail. A cross-reference is provided for all items.

1) ACCEPT statement (conversion of data). This item is required. This item shall be documented in the implementor's user documentation. (14.9.1, ACCEPT statement, general rules 1 and 14)

2) ACCEPT statement (device used when FROM is unspecified). This item is required. This item shall be documented in the implementor's user documentation. (14.9.1, ACCEPT statement, general rule 5)

3) ACCEPT statement, screen format (result when screen items overlap). This item is required. This item shall be documented in the implementor’s user documentation. (14.9.1, ACCEPT statement, general rule 22)

4) ACCEPT statement, screen format (when data is verified, behavior for inconsistent data). This item is required. This item shall be documented in the implementor's user documentation. (14.9.1, ACCEPT statement, general rule 20)

5) ACCEPT statement (size of data transfer). This item is required. This item shall be documented in the implementor's user documentation. (14.9.1, ACCEPT statement, general rule 2)

6) Alignment of alphanumeric group items (relative to first elementary item). This item is required. This item shall be documented in the implementor's user documentation. (8.5.1.5.1, Alignment of alphanumeric groups and of data items of usage display and 8.5.1.5.5, Alignment of strongly-typed group items)

7) Alignment of data for increased efficiency (special or automatic alignment: interpretation, implicit filler, semantics of statements). This item is optional. This item, if provided by the implementor, shall be documented in the implementor’s user documentation. (8.5.1.5.4, Item alignment for increased object-code efficiency; 13.18.54, SYNCHRONIZED clause, general rule 9)

8) ALPHABET clause (ordinal number of characters in the native coded character sets). This item is required. This item shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, general rule 6)

9) Alphanumeric literals (number of hexadecimal digits that map to an alphanumeric character). This item is required. This item shall be documented in the implementor's user documentation. (8.3.1.2.1, Alphanumeric literals, syntax rule 6)

10) Any-length elementary items (maximum length). This item is required. This item shall be documented in the implementor's user documentation. (8.5.1.7, Any-length elementary items)

11) Any-length elementary items (structure if neither the PREFIXED or DELIMITED is specified). This item is required. This item need not be documented in the implementor's user documentation. (8.5.1.7.1, Structure of an any-length elementary item)

12) ASSIGN clause, USING phrase (meaning and rules for operands; consistency rules). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.4.4, File control entry, general rule 4)

13) BACKGROUND-COLOR clause (the background color when the clause is not specified or the value specified is not in the range 0 to 7). This item is required. This item shall be documented in the implementor's user documentation. (13.18.4, BACKGROUND-COLOR clause, general rule 4)

14) Byte (number of bits in). This item is required. This item shall be documented in the implementor’s user documentation. (8.1.1, Computer's coded character set)

15) CALL statement (rules for program-name formation for a non-COBOL program). This item is conditionally required because it is conditioned upon support for calling non-COBOL programs. This item shall be documented in the implementor's user documentation. (14.9.4, CALL statement, general rule 3b)

16) CALL statement (runtime resources that are checked). This item is required. This item shall be documented in the implementor's user documentation. (14.9.4, CALL statement, general rule 3c)

17) CALL statement (rules for locating a non-COBOL program). This items is conditionally required because it is conditioned upon support for calling non-COBOL programs. This item shall be documented in the implementor's user documentation. (14.9.4, CALL statement, general rule 3b)

18) CALL statement (calling a non-COBOL program). This item is required. This item shall be documented in the implementor's user documentation. (14.9.4, CALL statement, general rule 3f)

19) CALL statement (other effects of the CALL statement). This item is required. This item shall be documented in the implementor's user documentation. (14.9.4, CALL statement, general rule 3g2)

20) CANCEL statement (result of canceling an active program when EC-PROGRAM-CANCEL-ACTIVE is not enabled). This item is required. This item shall be documented in the implementor's user documentation. (14.9.5, CANCEL statement, general rule 5)

21) CANCEL statement (result of canceling a non-COBOL program). This item is required. This item shall be documented in the implementor's user documentation. (14.9.5, CANCEL statement, general rule 10)

22) Case mapping. This item is required. This item shall be documented in the implementor's user documentation. (8.1.2, COBOL character repertoire, general rule 1)

    NOTE Implementors are strongly encouraged to define the case mapping according to the case mapping defined by the Unicode Consortium in the database UnicodeData.txt.

23) CHAR function (which one of the multiple characters is returned). This item is required. This item shall be documented in the implementor's user documentation. (15.14, CHAR function, returned values, rule 2)

24) CHAR-NATIONAL function (which one of the multiple characters is returned). This item is required. This item shall be documented in the implementor's user documentation. (15.15, CHAR-NATIONAL function, returned values, rule 2)

25) Characters prohibited from use in text-words in COPY … REPLACING and REPLACE statements. This item is required. This item shall be documented in the implementor's user documentation. (7.2.1.4, Text-words, rule 3).

26) CLOSE statement (closing operations). This item is required. This item shall be documented in the implementor's user documentation. (14.9.6, CLOSE statement, general rule 3c)

27) COBOL character repertoire (encoding of, mapping of, substitute graphics). This item is required. This item shall be documented in the implementor's user documentation. (8.1.2, COBOL character repertoire; 8.1.2, COBOL character repertoire, general rules 1 and 5)

28) COBOL character repertoire (if more than one encoding in a compilation group, control functions if any). Multiple encodings are optional and control functions are optional. This item is conditionally required because it is conditioned upon support of multiple encodings in a compilation group where control functions are defined for switching between encodings. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.1.2, COBOL character repertoire)

29) Color number (for a monochrome terminal, the mapping of the color attributes onto other attributes). This item is required. This item shall be documented in the implementor's user documentation. (9.2.7, Color number)

30) Compiler directives, compiler directive IMP (syntax rules and general rules). This item is conditionally required because it is conditioned upon support for an IMP directive. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (7.3, Compiler directives, syntax rule 9 and general rule 3)

31) Computer's coded character set (characters in and encoding of computer's alphanumeric coded character set and computer's national coded character set, encoding for usage DISPLAY and usage NATIONAL). This item is required. This item shall be documented in the implementor's user documentation. (8.1.1, Computer's coded character set; 8.3.1.2.1, Alphanumeric literals, syntax rule 2; 8.3.1.2.4, National literals, syntax rule 2)

32) Computer's coded character set (coded character values for certain COBOL items). This item is required. This item shall be documented in the implementor's user documentation. If values are determined at runtime, documentation shall specify the manner in which values are determined. (8.1.1, Computer's coded character set)

33) Computer's coded character set (correspondence between alphanumeric and national characters). This item is required. This item shall be documented in the implementor's user documentation. (14.9.24, MOVE statement, general rule 5; 8.8.4.1.1.6.2, Locale-based comparison; 15.23, DISPLAY-OF function, returned value rules 1 and 3; 15.61, NATIONAL-OF function, returned value rules 1 and 3)

34) Computer's coded character set (for literals, correspondence between compile-time and runtime character sets, when conversion takes place). This item is conditionally required because it is conditioned upon there being a difference between the compile-time and runtime computer's coded character sets. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.1.1, Computer's coded character set)

35) Computer's coded character set (correspondence between lowercase and uppercase letters when a locale is not in effect). This item is required. This item need not be documented in the implementor's user documentation. (15.53, LOWER-CASE function; 15.90, UPPER-CASE function)

36) Computer's coded character set (when composite alphanumeric and national, mapping of characters to each). This item is conditionally required because it is conditioned upon support for a composite alphanumeric and

national computer character set. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.1.1, Computer's coded character set)

37) Computer's coded character set (when more than one encoding, the mechanism for selecting encoding for runtime). This item is conditionally required because it is conditioned upon support for more than one encoding of a computer's coded character set. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.1.1, Computer's coded character set)

38) Computer's coded character set (whether UTF-8 or mixed alphanumeric and national characters recognized in class alphanumeric; applicable syntax and general rules). This item is conditionally required because it is conditioned upon support for UTF-8 or mixing alphanumeric and national characters in literals and data items of class alphanumeric. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.1.1, Computer's coded character set; 8.3.1.2.1, Alphanumeric literals, syntax rule 2 and general rule 3)

39) COPY statement (rules for identifying and locating default library text). This item is required. This item shall be documented in the implementor's user documentation. (7.2.2, COPY statement, syntax rule 5 and general rule 3)

40) CRT status 9xxx (the value of xxx for unsuccessful completion with implementor-defined conditions). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (9.2.3, CRT status)

41) Currency symbol (equivalence of non-COBOL characters). This item is required. This item shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, syntax rule 21)

42) Currency symbol (implementor-defined prohibition of non-COBOL characters). This item is required. This item shall be documented in the implementor's user documentation. (8.1.1, Computer's coded character sett)

43) Cursor (the cursor movement if keys are defined that change the cursor position). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (9.2.4, Cursor)

44) Data storage (possible representations when implementation provides multiple ways of storing data). This item is conditionally required because it is conditioned on whether the implementation provides multiple ways of storing data. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.5.1, Computer independent data description)

45) DEFINE directive (mechanism for providing value of a compilation-variable-name from the operating environment). This item is required. This item shall be documented in the implementor's user documentation. (7.3.9, DEFINE directive, general rule 4)

46) Devices that allow concurrent access. This item is required. This item shall be documented in the implementor's user documentation. (9.1.14, Sharing mode)

47) DISPLAY statement (data conversion). This item is required. This item shall be documented in the implementor's user documentation. (14.9.10, DISPLAY statement, general rule 1)

48) DISPLAY statement (format for display of a variable-length group). This item is required. This item shall be documented in the implementor's user documentation. (14.9.10, DISPLAY statement, general rule 7)

49) DISPLAY statement (size of data transfer). This item is required. This item shall be documented in the implementor's user documentation. (14.9.10, DISPLAY statement, general rule 2)

50) DISPLAY statement (standard display device). This item is required. This item shall be documented in the implementor's user documentation. (14.9.10, DISPLAY statement, general rule 8)

51) Dynamic-capacity table (determination of highest permissible occurrence number). This item is required. This item does not have to be documented in the implementor's documentation. (8.4.1.2, Subscripts)

52) Dynamic-capacity table (physical allocation). This item is required. This item does not have to be documented in the implementor's documentation. (8.5.1.6.3, Dynamic-capacity tables)

53) ENTRY-CONVENTION clause (entry-convention-names, their meanings and the default when not specified). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (11.9.6, ENTRY-CONVENTION clause, general rule 2)

54) EXIT and GOBACK statements (execution continuation in a non-COBOL runtime element). This item is conditionally required, because it is conditioned upon support for calling from non-COBOL programs. This item shall be documented in the implementor's user documentation. (14.9.13, EXIT statement, general rule 4; 14.9.17, GOBACK statement, general rule 1)

55) Exponentiation (results for certain operand values). This item is required. This item shall be documented in the implementor's user documentation. (8.8.1.3.6, Exponentiation, rule 2e)

56) External repository (mechanism for specifying whether checking and updating occur). This item is required. This item shall be documented in the implementor's user documentation. (8.13, External repository)

57) External repository information (other information beyond the required information). This item is optional. This item, if provided by the implementor, shall be documented in the implementor’s user documentation. (8.13, External repository)

58) Externalized names (formation and mapping rules). This item is required. This item shall be documented in the implementor’s user documentation. (7.3.8, CALL-CONVENTION directive, general rules 2a and 2b; 8.3.1.1.1, User-defined words)

59) Fatal exception condition (whether detected at compile time, circumstances under which detected). This item is optional. This item if provided by the implementor, does not have to be included in the implementor’s user documentation. (14.6.12.1.2, Fatal exception conditions)

60) Fatal exception condition (whether or not execution will continue, how it will continue, and how any receiving operands are affected when events that would cause a fatal exception to exist occur but checking for that condition is not enabled). This item is required. This item shall be documented in the implementor’s user documentation. (14.6.12.1.2, Fatal exception conditions)

61) FILE-CONTROL entry, ASSIGN clause (TO phrase meaning and rules). This item is required. This item shall be documented in the implementor's user documentation. (12.4.4, File control entry, syntax rules 5 and 6)

62) FILE-CONTROL entry, ASSIGN clause (consistency rules for external file connectors). This item is required. This item shall be documented in the implementor's user documentation. (12.4.4, File control entry, general rule 1)

63) FILE-CONTROL entry, ASSIGN clause (USING phrase meaning and rules). This item is required. This item shall be documented in the implementor's user documentation. (12.4.4, File control entry, general rule 4)

64) Figurative constant values (representation of zero, space, and quote). This item is required. This item shall be documented in the implementor's user documentation. (8.3.1.2.5, Figurative constant values)

65) File sharing (interaction with other facilities and languages). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (9.1.14, Sharing mode)

66) File sharing (which devices allow concurrent access to the file). This item is required. This item shall be documented in the implementor’s user documentation. (9.1.14, Sharing mode)

67) File sharing (default mode when unspecified). This item is required. This item shall be documented in the implementor's user documentation. (9.1.3, File connector; 9.1.14, Sharing mode; 14.9.26, OPEN statement, general rule 22)

68) Fixed file attribute (whether the ability to share a file is a fixed file attribute). This item is required. This item shall be documented in the implementor's user documentation. (9.1.6, Fixed file attributes)

69) FLAG-02 directive (warning mechanism). This item is required. This item shall be documented in the implementor's user documentation. (7.3.11, FLAG-02 directive, general rule 1)

70) FLAG-85 directive (warning mechanism). This item is required. This item shall be documented in the implementor's user documentation. (7.3.12, FLAG-85 directive, general rule 1)

71) Floating-point numeric item (alignment when used as a receiving operand). This item is required. This alignment does not have to be documented in the implementor's user documentation. (14.6.8, Alignment of data within data items, rule 2)

72) Floating-point numeric literals (maximum permitted value and minimum permitted value of the exponent). This item is required. This item shall be documented in the implementor's user documentation. (8.3.1.2.2.2, Floating-point numeric literals, rule 3)

73) FOREGROUND-COLOR clause (the foreground color when the clause is not specified or the value specified is not in the range 0 to 7). This item is required. This item shall be documented in the implementor's user documentation. (13.18.22, FOREGROUND-COLOR clause, general rule 4)

74) FORMAT clause (representation produced). This item is required. This item shall be documented in the implementor's user documentation. (13.18.23, FORMAT clause, general rule 10)

75) FORMAT clause (exclusions on restoring to same internal representation). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (13.18.23, FORMAT clause, general rule 11)

76) Format validation (rules for checking items of usages other than display or national). This item is required. This item shall be documented in the implementor's user documentation. (13.18.39, PICTURE clause, general rule 14; 13.18.59, USAGE clause, general rule 3)

77) FORMATTED-CURRENT-DATE (accuracy of returned time). This item is required. This item shall be documented in the implementor's user documentation. (15.34, FORMATTED-CURRENT-DATE function, returned value rule 1).

78) Function-identifier (execution of a non-COBOL function when a function-prototype-name is specified). This item is conditionally required because it is conditioned upon support for calling non-COBOL functions. This item shall be documented in the implementor's user documentation. (8.4.2.2, Function-identifier, general rule 6d)

79) Function-identifier (object time resources that are checked). This item is required. This item shall be documented in the implementor's user documentation. (8.4.2.2, Function-identifier, general rule 6c)

80) Function-identifier (result when argument rules are violated and checking for the EC-ARGUMENT-FUNCTION exception condition is not enabled). This item is required. The value returned does not have to be documented in the implementor's user documentation. (15.3, Arguments)

81) Function keys (context-dependent keys, function number, and method for enabling and disabling). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (9.2.2, Function keys)

82) Function returned values (characteristics, representation, and returned value for native arithmetic). This item is required. The value returned does not have to be documented in the implementor's user documentation. (15.4.1, Numeric and integer functions)

83) Hexadecimal alphanumeric literals (mapping for non-existing corresponding character). This item is required. This item shall be documented in the implementor's user documentation. (8.3.1.2.1, Alphanumeric literals, general rule 4)

84) Hexadecimal alphanumeric literals (mapping when characters not multiples of four bits). This item is conditionally required because it is conditioned upon a computer's coded character set with characters that are not multiples of four bits. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.3.1.2.1, Alphanumeric literals, general rule 4)

85) Hexadecimal national literals (mapping for non-existing corresponding character). This item is required. This item shall be documented in the implementor's user documentation. (8.3.1.2.4, National literals, general rule 4)

86) Hexadecimal national literals (mapping when characters not multiples of four bits). This item is conditionally required because it is conditioned upon a computer's coded character set with characters that are not multiples of four bits. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.3.1.2.4, National literals, general rule 4)

87) Implementor-defined exception conditions, EC-xxx-IMP and EC-IMP-xxx (specification and meaning of xxx). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (Table 14, Exception-names and exception conditions)

88) Implementor-defined level-2 exception conditions, EC-level-2-IMP (specification and meaning of the specified level-2 exception condition). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (Table 14, Exception-names and exception conditions)

89) INVOKE statement (behavior when invoking a non-COBOL method). This item is required. This item shall be documented in the implementor's user documentation. (14.9.22, INVOKE statement, general rule 2b)

90) INVOKE statement (runtime resources that are checked). This item is required. This item shall be documented in the implementor's user documentation. (14.9.22, INVOKE statement, general rule 7d)

91) I-O status (action taken for fatal exception conditions). This item is required. This item shall be documented in the implementor's user documentation. (9.1.12, I-O status)

92) I-O status (if more than one value applies). This item is required. The internal procedure employed by the implementor does not have to be documented in the implementor's user documentation. (9.1.12, I-O status)

93) I-O status, permanent error (technique for error correction). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (9.1.12, I-O status)

94) I-O status 0x (value of x). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (9.1.12.2, Implementor-defined successful completion)

95) I-O status 24 (manner in which the boundaries of a file are defined). This item is required. This item shall be documented in the implementor's user documentation. (9.1.12.4, Invalid key condition with unsuccessful completion)

96) I-O status 34 (manner in which the boundaries of a file are defined). This item is required. This item shall be documented in the implementor's user documentation. (9.1.12.5, Permanent error condition with unsuccessful completion)

97) I-O status 52 (conditions under which deadlock is detected). This item is required. This item shall be documented in the implementor's user documentation. (9.1.12.7, Record operation conflict condition with unsuccessful completion)

98) I-O status 9x (value of x). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (9.1.12.9, Implementor-defined condition with unsuccessful completion)

99) LEAP-SECOND directive (whether a value greater than 59 seconds may be reported and, if so, the maximum number of seconds that may be reported). This item is required. This item shall be documented in the implementor's user documentation. (7.3.15, LEAP-SECOND directive, general rule 2; 14.9.1, ACCEPT statement, general rule 11; 15.18, CURRENT-DATE function, returned value rule 1; 15.92, WHEN-COMPILED function, return value rule 1)

100) LEAP-SECOND directive (whether standard numeric time form values greater than or equal to 86,400 may be reported). This item is required. This item shall be documented in the implementor's user documentation. (7.3.15, LEAP-SECOND directive, general rule 4; 15.75, SECONDS-PAST-MIDNIGHT function, returned value rule 4, 15.3.1.2.1.3, Permissible values for data associated with common time formats)

101) Life cycle for objects (timing and algorithm for taking part in continued execution). This item is required. This item need not be documented in the implementor's user documentation. (9.3.14.2, Life cycle for instance objects)

102) Linkage section (whether access to linkage section items is meaningful when called from a non-COBOL program). This item is conditionally required because it is conditioned upon support for calling a COBOL program from a non-COBOL program. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (13.7, Linkage section)

103) Listings (whether and when produced by the compiler, effect of logical conversion). This item is required. This item shall be documented in the implementor's user documentation. (6.5, Logical conversion; 7, Compiler directing facility; 7.3.16, LISTING directive, general rule 1)

104) Locale specification (how user and system defaults defined; at least one user and one system default). This item is required. This item need not be documented in the implementor's user documentation. The implementor may specify that the user locale and the system locale are the same locale. (8.2, Locales)

105) Locale specification (manner of implementation). This item is required because at least one default locale is required. This item does not have to be documented in the implementor's user documentation. (8.2, Locales)

106) Locale switch (whether a switch by a non-COBOL runtime module is recognized by COBOL). This item is conditionally required because it is conditioned upon support for activation of non-COBOL runtime modules. This item, if provided, shall be documented in the implementor's user documentation. (8.2, Locales)

107) METHOD-ID paragraph (actual method-name used when PROPERTY phrase is specified). This item is required. This item shall be documented in the implementor's user documentation. (11.7, METHOD-ID paragraph, general rule 1)

108) National literals (number of hexadecimal digits that map to a national character). This item is required. This item shall be documented in the implementor's user documentation. (8.3.1.2.4, National literals, syntax rule 5)

109) Native arithmetic (techniques used, intermediate data item). This item is required. The internal procedure employed by the implementor does not have to be documented in the implementor's user documentation. (8.8.1.2, Native arithmetic; 11.9.4, ARITHMETIC clause; 14.7.6, Arithmetic statements)

110) Native arithmetic (when an operand or arithmetic expression is an integer). This item is required. This item shall be documented in the implementor's user documentation. (5.5, Integer operands)

111) NULL (value of NULL). This item is required. This item need not be documented in the implementor's user documentation. (8.4.2.7, NULL object reference, general rule 1; 8.4.2.10, NULL address pointer, General rules 1, 2, and 3)

112) OBJECT-COMPUTER paragraph (default object computer). This item is required. This item shall be documented in the implementor's user documentation. (12.3.5, OBJECT-COMPUTER paragraph, general rules 3 and 4)

113) OBJECT-COMPUTER paragraph (computer-name and implied equipment configuration). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.3.5, OBJECT-COMPUTER paragraph, general rule 2)

114) OCCURS clause (range of values allowed in the index). This item is required. This item shall be documented in the implementor's user documentation. (13.18.37, OCCURS clause, general rule 2)

115) OPEN statement (validation of fixed file attributes). This item is required. This item shall be documented in the implementor's user documentation. (14.9.26, OPEN statement, general rule 8)

116) OPEN statement with OUTPUT phrase (positioning of the output file with regard to physical page boundaries). This item is required. This item shall be documented in the implementor's user documentation. (14.9.26, OPEN statement, general rule 16)

117) OPEN statement without the SHARING phrase and no SHARING clause in the file control entry (definition of sharing mode established for each file connector). This item is required. This item shall be documented in the implementor's user documentation. (14.9.26, OPEN statement, general rule 22)

118) Ordering table (allowable content of literal identifying an ordering table that complies with ISO/IEC 14651). This item is conditionally required because it is conditioned upon support of ISO/IEC 14651. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, general rule 17)

119) Parameterized classes and interfaces (when expanded). This item is required. This item need not be documented in the implementor's user documentation. (7, Compiler directing facility)

120) Procedure division header rules when either the activating or the activated runtime element is not a COBOL element (restrictions and mechanisms for all supported language products with details such as the matching of parameters, data type representation, returning of a value, and omission of parameters). This item is conditionally required because it is conditioned upon support for a language other than COBOL. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (14, Procedure division, general rule 13)

121) Program-address identifier (relation between address and non-COBOL program). This item is conditionally required because it is conditioned upon support of non-COBOL programs. This item shall be documented in the implementor's user documentation. (8.4.2.13, Program-address-identifier, general rule 2)

122) Program-name (formation rules for a non-COBOL program). This item is optional. This item if provided by the implementor, shall be documented in the implementor's user documentation. (8.3.1.1.1.20, Program-name)

123) RANDOM function (seed value when no argument on first reference). This item is required. The seed value used by the implementor does not have to be documented in the implementor's user documentation. (15.70, RANDOM function, argument rule 4)

124) RANDOM function (subset of the domain of argument-1). This item is required. This item shall be documented in the implementor's user documentation. (15.70, RANDOM function, returned values, rule 3)

125) RECORD clause (calculations to derive size of records on storage medium). This item is required. This item shall be documented in the implementor's user documentation. (13.18.42, RECORD clause, general rule 2)

126) RECORD clause (implicit RECORD clause if RECORD clause is not specified). This item is required. This item shall be documented in the implementor's user documentation. (13.18.42, RECORD clause, general rule 5)

127) RECORD clause (whether fixed or variable records produced for fixed-or-variable-length format). This item is required. This item shall be documented in the implementor's user documentation. (13.18.42, RECORD clause, general rule 17)

128) RECORD DELIMITER clause (consistency rules when used with external file connectors). This item is conditionally required because it is conditioned upon the existence of the optional feature-name in the RECORD DELIMITER clause. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.4.4, File control entry, general rule 1c)

129) RECORD DELIMITER clause (feature-name and associated method for determining length of variable-length records). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.4.4.10, RECORD DELIMITER clause, syntax rule 2 and general rule 4)

130) RECORD DELIMITER clause (if not specified, method for determining length of variable-length records). This item is required. This item shall be documented in the implementor's user documentation. (12.4.4.10, RECORD DELIMITER clause, general rule 5)

131) Record locking (circumstances other than a locked logical record that return a locked record status). This item is conditionally required because it is conditioned on the existence of such circumstances in an implementation. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (9.1.15, Record locking)

132) Record locking (default mode when unspecified by user). This item is required. This item shall be documented in the implementor's user documentation. (12.4.4.8, LOCK MODE clause, general rule 1)

133) Record locking (maximum number allowed for a run unit). This item is required and shall be at least 255. This item shall be documented in the implementor's user documentation. (12.4.4.8, LOCK MODE clause, general rule 7).

134) Record locks (maximum number allowed for a file connector). This item is required and shall be at least 15. This item shall be documented in the implementor's user documentation. (12.4.4.8, LOCK MODE clause, general rule 7)

135) Reference format (control characters in a free-form line). This item is required. This item shall be documented in the implementor's user documentation. (6, Reference format, rule 3b)

136) Reference format (meaning of lines and character positions in free-form and fixed-form format). This item is required. This item shall be documented in the implementor's user documentation. (6, Reference format, rule 1c)

137) Reference format (rightmost character position of program-text area). This item is required. This item shall be documented in the implementor's user documentation. (6.3, Fixed-form reference format, margin R.)

138) Report file (record structure). This item is required. This item shall be documented in the implementor's user documentation. (13.4.4, File description entry, general rule 4)

139) Report writer printable item (fixed correspondence between columns and national characters). This item is required. This item shall be documented in the implementor's user documentation. (13.18.14, COLUMN clause, general rule 2)

140) REPOSITORY paragraph (how external repository and class-specifier determine which class is used). This item is required. This item shall be documented in the implementor's user documentation. (12.3.7, REPOSITORY paragraph, general rule 6)

141) REPOSITORY paragraph, INTERFACE phrase (how interface specifier and external repository determine which interface is used). This item is required. This item shall be documented in the implementor's user documentation. (12.3.7, REPOSITORY paragraph, general rule 9)

142) REPOSITORY paragraph (when the AS phrase is required). This item is required. This item shall be documented in the implementor's user documentation. (12.3.7, REPOSITORY paragraph, general rule 2)

143) RESERVE clause (number of input-output areas, if not specified). This item is required. This item shall be documented in the implementor's user documentation. (12.4.4.13, RESERVE clause, general rule 1)

144) RETRY phrase (interval between attempts to obtain access to a locked file or record). This item is required. This item shall be documented in the implementor's user documentation. (14.7.8, RETRY phrase, general rule 1)

145) RETRY phrase (maximum meaningful time-out value and internal representation; technique for determining frequency of retries). This item is required. This item shall be documented in the implementor's user documentation. (14.7.8, RETRY phrase, general rule 2)

146) Run unit (relationship and interaction with non-COBOL components). This item is conditionally required because it is conditioned upon support for non-COBOL modules. This item shall be documented in the implementor's user documentation. (14.6.1, Run unit organization).

147) Run unit termination (whether locale reset). This item is required. This item shall be documented in the implementor's user documentation. (14.6.10, Normal run unit termination)

148) SAME SORT/SORT-MERGE AREA clause (extent of allocation). This item is required. The internal procedure employed by the implementor does not have to be documented in the implementor's user documentation. (12.4.6, SAME clause, general rule 3b)

149) SEARCH ALL statement (varying of the search index during the search operation). This item is required. The internal procedure employed by the implementor does not have to be documented in the implementor's user documentation. (14.9.34, SEARCH statement, general rule 9)

150) SECONDS-PAST-MIDNIGHT function returned value (precision). This item is required. This item shall be documented in the implementor's user documentation. (15.75, SECONDS-PAST-MIDNIGHT function, returned value rule 3)

151) SECURE clause (cursor movement when data is entered into a field for which the SECURE clause is specified). This item is item is required. This item shall be documented in the implementor's user documentation. (13.18.49, SECURE clause, general rule 4)

152) SELECT WHEN clause (whether a SELECT WHEN takes effect for READ statements and REWRITE or WRITE statements with the FILE phrase in the absence of a CODE-SET clause or a FORMAT clause). This item is optional. If provided, this item shall be documented in the implementor's user documentation. (13.18.50, SELECT WHEN clause, general rule 5)

153) SET statement (effect of SET on function whose address is being stored in a function-pointer). This item is required. This item shall be documented in the implementor's user documentation. (14.9.35, SET statement, general rule 15)

154) SET statement (effect of SET on program whose address is being stored in a program-pointer). This item is required. This item shall be documented in the implementor's user documentation. (14.9.35, SET statement, general rule 17)

155) SIGN clause (representation when PICTURE contains character 'S' with no optional SIGN clause). This item is required. This item shall be documented in the implementor's user documentation. (13.18.51, SIGN clause, general rule 4 and 8.5.1.4, Algebraic signs)

156) SIGN clause (valid sign when SEPARATE CHARACTER phrase not present). This item is required. This item shall be documented in the implementor's user documentation. (13.18.51, SIGN clause, general rule 5b)

157) Size error condition (whether or not range of values allowed for the intermediate data item is to be checked). This item is required. This item shall be documented in the implementor's user documentation. (14.7.4, SIZE ERROR phrase and size error condition)

158) SPECIAL-NAMES paragraph (allowable locale-names and literal values). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, general rule 5)

159) SPECIAL-NAMES paragraph, ALPHABET clause (coded character set referenced by STANDARD-2 phrase). This item is required. This item shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, general rule 7c)

160) SPECIAL-NAMES paragraph, ALPHABET clause, code-name-1 (alphanumeric coded character set and collating sequence; ordinal number of characters; correspondence with native alphanumeric character set). This item is conditionally required because it is conditioned upon the existence of a code-name. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, general rule 7i)

161) SPECIAL-NAMES paragraph, ALPHABET clause, code-name-1 and code-name-2 (the names supported for code-name-1 and code-name-2). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph syntax rule 15)

162) SPECIAL-NAMES paragraph, ALPHABET clause, code-name-2 (national coded character set and collating sequence; ordinal number of characters; correspondence with native national character set). This item is conditionally required because it is conditioned upon the existence of a code-name. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, general rule 7j)

163) SPECIAL-NAMES paragraph, ALPHABET clause, literal phrase (ordinal number of characters not specified). This item is required. The implementor shall document the scheme used for assigning the ordinal number, but does not need to specify the ordinal number character-by-character. (12.3.6, SPECIAL-NAMES paragraph general rule 7k4)

164) SPECIAL-NAMES paragraph, ALPHABET clause, STANDARD-1 and STANDARD-2 phrases (correspondence with native character set). This item is required. This item shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, general rule 7c)

165) SPECIAL-NAMES paragraph, ALPHABET clause, UCS-4, UTF-8, and UTF-16 phrases (correspondence with native character set). This item is required. This item shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, general rules 7f and 7h)

166) SPECIAL-NAMES paragraph, device-name (names available, restrictions on use). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, syntax rules 7 and 8)

167) SPECIAL-NAMES paragraph, feature-name (names available, any positioning rules, any restrictions on use). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, syntax rules 6 and 8)

168) SPECIAL-NAMES paragraph, switch-name (names available, which switches may be referenced by the SET statement, scope of, and external facility for modification). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (12.3.6, SPECIAL-NAMES paragraph, syntax rule 8 and general rules 2 and 3)

169) Standard intermediate data item (representation). This item is required. This item shall be documented in the implementor's user documentation. (8.8.1.3.1, Standard intermediate data item)

170) STOP statement (constraints on the value of the STATUS literal or on the contents of the data item referenced by the STATUS identifier). This item is required. This item shall be documented in the implementor's user documentation. (14.9.38, STOP statement, general rule 5)

171) STOP statement (mechanism for error termination). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (14.9.38, STOP statement, general rule 4)

172) Subscripts (mapping indexes to occurrence numbers). This item is required. The internal procedure employed by the implementor does not have to be documented in the implementor's user documentation. (8.4.1.2, Subscripts, general rule 1c)

173) Switch-name (Identifies an external switch). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.3.1.1.2.10, Switch-name)

174) SYNCHRONIZED clause (effect on elementary items and containing records or groups; implicit filler generation). This item is required. This item shall be documented in the implementor's user documentation. (8.5.1.5.3, Alignment of data items of usage bit; 13.18.54, SYNCHRONIZED clause, general rule 8)

175) SYNCHRONIZED clause (how records of a file are handled). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (13.18.54, SYNCHRONIZED clause, general rule 10)

176) SYNCHRONIZED clause (positioning when neither RIGHT or LEFT is specified). This item is required. This item shall be documented in the implementor's user documentation. (13.18.54, SYNCHRONIZED clause, general rule 2)

177) System-names (rules for formation of a system-name). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (8.3.1.1.2, System-names)

178) Terminal screen (correspondence of a column and a character in the computer's national coded character set). This item is required. This item shall be documented in the implementor's user documentation. (9.2.1, Terminal screen)

179) Text manipulation (stage of processing the LISTING and PAGE directives and the SUPPRESS phrase of COPY). This item is conditional. This item is conditioned on the implementor producing listings. This item, if provided, need not be documented in the implementor's user documentation. (7.2, Text manipulation)

180) Text manipulation (stage of processing parameterized class expansion). This item is required. This item need not be documented in the implementor's user documentation. (7.2, Text manipulation)

181) Time formats and corresponding function values (maximum precision not less than nine fractional digits). This item is required. This item shall be documented in the implementor's user documentation. (15.3.1.2.1.2, Common time formats with fractional seconds representation)

182) THROUGH phrase in VALUE clause and EVALUATE statement (collating sequence used for determining range of values when no alphabet-name is specified). This item is required. This item shall be documented in the implementor's user documentation. (14.7.7, THROUGH phrase, rule 2)

183) TURN directive (whether location information is available when the LOCATION phrase is not specified). This item is required. This item shall be documented in the implementor's user documentation. (7.3.20, TURN directive, general rule 6)

184) USAGE BINARY clause (computer storage allocation, alignment and representation of data). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 4)

185) USAGE BINARY-CHAR, BINARY-SHORT, BINARY-LONG, BINARY-DOUBLE (allow wider range than minimum specified). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 11)

186) USAGE BINARY-SHORT, BINARY-LONG. BINARY-DOUBLE, FLOAT-SHORT, FLOAT-LONG, FLOAT-EXTENDED (representation and length of data item associated with). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 18)

187) USAGE COMPUTATIONAL clause (alignment and representation of data). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 6)

188) USAGE DISPLAY (size and representation of characters). This item is required; an implementor may provide an option to vary the size or representation for different compilation units. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 7)

189) USAGE FLOAT-SHORT, FLOAT-LONG, FLOAT-EXTENDED (size and permitted range of value). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 12)

190) USAGE FUNCTION-POINTER clause (alignment, size, and representation of data; and allowable languages). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 23)

191) USAGE INDEX clause (alignment and representation of data). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 9)

192) USAGE NATIONAL (size and representation of characters). This item is required; an implementor may provide an option to vary the size or representation for different compilation units. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 8)

193) USAGE OBJECT REFERENCE clause (amount of storage allocated). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause general rule 19a)

194) USAGE PACKED-DECIMAL clause (computer storage allocation, alignment and representation of data). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 10)

195) USAGE POINTER clause (alignment, size, representation, and range of values). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 20)

196) USAGE PROGRAM-POINTER clause (alignment, size, and representation of data; and allowable languages). This item is required. This item shall be documented in the implementor's user documentation. (13.18.59, USAGE clause, general rule 21)

197) USE statement (action taken following execution of the USE procedure when I-O status value indicates a fatal exception condition). This item is required. This item shall be documented in the implementor's user documentation. (14.9.45, USE statement, general rules 7 and 12; 9.1.12, I-O status)

198) User-defined words (whether extended letters may be specified in user-defined words externalized to the operating environment). This item is required. This item shall be documented in the implementor's user documentation. (8.3.1.1.1, User-defined words)

199) Variable-length data items (actual time when the resources used are freed). This item is required. This item does not have to be documented in the implementor's documentation. (8.5.1.8.2, Availability and persistence of data items)

200) WRITE statement (mnemonic-name-1). This item is optional. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (14.9.47, WRITE statement, syntax rule 16)

201) WRITE statement (page advance when mnemonic-name-1 specified). This item is conditionally required because it is conditioned upon the existence of a mnemonic-name-1 for the WRITE statement. This item, if provided by the implementor, shall be documented in the implementor's user documentation. (14.9.47, WRITE statement, general rule 22d)

A.2 Undefined language element list

The following is a list of the COBOL language elements within this final committee draft International Standard that are explicitly undefined.

1) ALLOCATE statement. If the INITIALIZED phrase is not specified in the ALLOCATE statement and arithmetic-expression-1 is specified, the content of allocated storage is undefined. (14.9.3, ALLOCATE statement, general rule 8)

2) ALLOCATE statement. If the INITIALIZED phrase is not specified in the ALLOCATE statement and data-name-1 is specified, the content of allocated storage is undefined except for data items of class object or class pointer. (14.9.3, ALLOCATE statement, general rule 9)

3) CALL statement. If a program-pointer data item containing an invalid program address is used in a CALL statement, execution of the CALL statement is undefined. (14.9.4, CALL statement, general rule 3f)

4) CANCEL statement. If a program-pointer has been set to point to the program to be canceled, the result of referencing that program-pointer in a subsequent CALL statement is undefined. (14.9.5, CANCEL statement, general rule 11)

5) CLOSE statement. The unsuccessful execution of the CLOSE statement without the UNIT phrase leaves the availability of the record area undefined. (14.9.6, CLOSE statement, general rule 7)

6) COLUMN clause. The result of printing a report line is undefined if any given column position in the report line is occupied by more than one printable item when the line is printed. (13.18.14, COLUMN clause, general rule 4)

7) Cursor. The position and visibility of the cursor is undefined during execution of a DISPLAY screen statement. (9.2.4, Cursor)

8) Cursor locator. The value of the cursor locator is undefined after an unsuccessful execution of an ACCEPT screen statement. (9.2.5, Cursor locator)

9) Cursor locator. The value of the cursor locator is undefined if the position of the visible cursor is at a line or column number that is greater than 999 when the execution of an ACCEPT screen statement is terminated. (9.2.5, Cursor locator)

10) Exception conditions. The exception condition that is set to exist if multiple exceptions are detected during the execution of a statement is undefined unless otherwise specified. (14.6.12.1, Exception conditions)

11) Explicit and implicit transfers of control. In the declarative section the flow of control is undefined when there is no next executable statement after either the last statement when the paragraph in which it appears is not being executed under the control of some other COBOL statement, or the last statement when the statement is in the range of an active PERFORM statement executed in a different section and this last statement of the declarative section is not also the last statement of the procedure that is the exit of the active PERFORM statement. (14.6.3, Explicit and implicit transfers of control)

12) File section. The initial value of the data items in the file section is undefined. (13.4, File section, general rule 1 and 13.18.62, VALUE clause, general rule 2)

13) FREE statement. The contents of any data items located within the dynamic storage area being released become undefined. (14.9.14, FREE statement, general rule 1)

14) Function-identifier. The results of the execution of a function are undefined if the activated function references an omitted argument. (8.4.2.2, Function-identifier, general rule 8)

15) Incompatible data. When a numeric-edited data item is the sending operand of a de-editing MOVE statement and the content of that data item is not a possible result for any editing operation in that data item, the result of the MOVE operation is undefined and an EC-DATA-INCOMPATIBLE exception condition is set to exist. (14.6.12.2, Incompatible data)

16) Incompatible data. When the content of a boolean or numeric sending operand is referenced during the execution of a statement and the content of that sending operand would result in a false value in the corresponding class test, the result of the reference is undefined and an EC-DATA-INCOMPATIBLE exception condition is set to exist, except when the sending operand is referenced in a class condition or in a VALIDATE statement. (14.6.12.2, Incompatible data)

17) Incompatible data on partial reference. If part of a sending operand's content is referenced by a given execution of a statement, it is undefined whether any incompatible data in the unreferenced content is detected. (14.6.12.2, Incompatible data)

18) Initial items. It is undefined whether each activation of an initial program has its own copy of initial items. (8.6.3, Automatic, initial, and static internal items)

19) Initial state. If a VALUE clause is not associated with a data item described in the working-storage section or local-storage section with the exception of data items of category object reference and data items of class pointer, the initial value of the data item is undefined. Also, the initial value of the following are undefined: an index, a based item, and an item described with the EXTERNAL clause. (13.18.62, VALUE clause, general rule 4)

20) INITIALIZE statement. The result of the execution of the INITIALIZE statement is undefined if the receiving operand and the sending operand occupies the same storage area even if they are defined by the same data description entry. (14.9.19, INITIALIZE statement, general rule 8)

21) INSPECT statement. The results of the execution of an INSPECT statement are undefined if:

    a) the CONVERTING phrase is specified and the size of the item preceding TO is not equal to the size of the item following TO;

    b) the CHARACTERS phrase is specified and the size of the replacement item is not one character;

    c) replacing-phrase is specified and CHARACTERS is not specified and the size of the replacing item is not equal to the size of the item being replaced;

    d) the INSPECT identifier, ALL identifier, LEADING identifier, AFTER identifier, or BEFORE identifier occupies the same storage area as the TALLYING identifier;

    e) the CHARACTERS BY identifier, ALL identifier, LEADING identifier, FIRST identifier, BY identifier, AFTER identifier, or BEFORE identifier occupies the same storage area as the INSPECT identifier;

    f) the CONVERTING identifier, TO identifier, AFTER identifier, or BEFORE identifier occupies the same storage area as the INSPECT identifier.

    (14.9.21, INSPECT statement, general rules 13, 14, 15, 18, 21, and 22)

22) Intrinsic functions. The evaluation of an ALL subscript shall result in at least one argument, otherwise the result of the reference to the function-identifier is undefined. (15.3, Arguments)

23) Linkage section. If a formal parameter or returning item in the linkage section is accessed in a program that is not a called program, such as a program that is activated by the operating system, the effect is undefined. (13.7, Linkage section, general rule 3)

24) Linkage section. If the runtime element containing the linkage section is activated by the operating system, the initial value of a linkage section data item is undefined. (13.7, Linkage section, general rule 5)

25) LINE clause. The results of printing a report that is divided into pages are undefined if each report group does not fit on one page. (13.18.34, LINE clause, general rule 2)

26) LINE clause. The results of printing a report group are undefined if any lines or groups of lines overlap each other, except that the nonspace characters of a relative line specified with a relative line number of zero will overwrite the corresponding characters of the preceding line. (13.18.34, LINE clause, general rule 3)

27) MERGE statement. The results of the merge operation are undefined if the records in the USING files are not ordered as described in the ASCENDING or DESCENDING KEY phrases in the collating sequence associated with the MERGE statement. (14.9.23, MERGE statement, general rule 6)

28) MERGE statement. For a relative file, the content of the relative key data item is undefined after the execution of the MERGE statement. (14.9.23, MERGE statement, general rule 7c)

29) MERGE statement. The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the file description entry for the USING file or files is undefined upon completion of the MERGE statement. (14.9.23, MERGE statement, general rule 7)

30) MERGE statement. The results of the execution of a MERGE statement are undefined if the range of the output procedure causes the execution of any MERGE, RELEASE, or file format SORT statement. (14.9.23, MERGE statement, general rule 8)

31) MERGE statement. The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the sort-merge file description entry for the MERGE file is undefined upon completion of the MERGE statement for which the GIVING phrase is specified. (14.9.23, MERGE statement, general rule 12)

32) MOVE statement. The results of the execution of a MOVE statement are undefined if the contents of the sending operand are not valid according to the rules for incompatible data. (14.9.24, MOVE statement, general rules 5g and 5i, and 14.6.12.2, Incompatible data)

33) MOVE statement. The result of the execution of a MOVE statement is undefined if the sending operand is of category alphanumeric-edited or national-edited and the receiving operand shares the same storage area and the same data-description entry as the sending operand. (14.9.24, MOVE statement, General rule 5b).

34) OCCURS clause. The value of an index is undefined when execution of a statement creates a value for the index that is outside the range of the values allowed by the implementor. (13.18.37, OCCURS clause, general rule 2)

35) OCCURS clause, DEPENDING phrase. For an occurs-depending table, the content of a data item whose occurrence number exceeds the value of the data item referenced by the DEPENDING ON data-name is undefined. (13.18.37, OCCURS clause, general rule 7)

36) Overlapping operands. The situations where the results of an operation involving overlapping operands are undefined as follows:

a) When a sending and a receiving operand in any statement share a part or all of their storage areas, yet are not defined by the same data description entry, and the sequence of operations described in the rules for the statement do not ensure a defined result.

b) When one or more of the operands is reference modified.

(14.6.9, Overlapping operands)

37) PERFORM statement. The results of the execution of overlapping PERFORM statements are undefined. (14.9.27, PERFORM statement, general rule 11)

38) Procedure division header. The Initial value of the RETURNING data-name is undefined. (14.2, Procedure division structure, general rule 7)

39) READ statement. The content of the associated record area is undefined at the completion of most unsuccessful executions of the READ statement. (14.9.29, READ statement, general rules 10c and 16)

40) READ statement. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement. (14.9.29, READ statement, general rule 3)

41) READ statement. The key of reference is undefined at the completion of most unsuccessful executions of the READ statement for an indexed file. (14.9.29, READ statement, general rule 16)

42) READ statement. The portion of the record area that is to the right of the last valid character read is undefined when the number of bytes in the record that is read is less than the minimum size specified by the record description entries for the file being read. (14.9.29, READ statement, general rule 14)

43) RECORD clause. The contents of any implicit filler bits generated to complete the last byte of a logical record are undefined. (13.18.42, RECORD clause, general rule 4)

44) RELEASE statement. The content of the bytes that extend beyond the end of the record are undefined when the number of bytes to be released to the sort operation is greater than the number of bytes in the record. (14.9.30, RELEASE statement, general rule 6)

45) RETURN statement. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the RETURN statement. (14.9.32, RETURN statement, general rule 2)

46) RETURN statement. The contents of the record area are undefined when the at end condition occurs. (14.9.32, RETURN statement, general rule 3)

47) RETURN statement. The results of the execution of a RETURN statement are undefined if an attempt is made to execute a RETURN statement as part of the current output procedure after an at end condition occurs. (14.9.32, RETURN statement, general rule 3)

48) REWRITE statement. The content of the bytes that extend beyond the end of the record are undefined when the number of bytes to be written to the file is greater than the number of bytes in the record. (14.9.33, REWRITE statement, general rule 15)

49) SEARCH ALL statement. If one or more settings of the search index satisfy all conditions in the WHEN phrase, it is undefined whether the search will succeed in identifying any applicable index unless:

a) The contents of each key data item referenced in the WHEN phrase are sequenced in the table according to the ASCENDING or DESCENDING phrase associated with that key data item, and

b) When the table is subordinate to one or more data description entries that contain an OCCURS clause, the evaluation of the conditions within a WHEN phrase that references a key data item subordinate to the table

results in the same occurrence number for any subscripts associated with a given level of the superordinate tables.

(14.9.34, SEARCH statement, general rule 6)

50) SEARCH ALL statement. If there is more than one setting of the search index that satisfies all conditions in the WHEN phrase, it is undefined which one will be used as the final setting of the search index. (14.9.34, SEARCH statement, general rule 7)

51) SEARCH ALL statement. If any of the conditions specified in the WHEN phrase are not satisfied for any setting of the index within the permitted range, control is passed to the AT END phrase imperative statement, when specified, or to the end of the SEARCH statement when the AT END phrase is not specified; in either case the final setting of the search index is undefined. (14.9.34, SEARCH statement, general rule 9)

52) SORT statement. If the DUPLICATES phrase is not specified and the contents of all the key data items associated with one data record or table element are equal to the contents of the corresponding key data items associated with one or more other data records or table elements, the order of return of these records or the relative order of the contents of these table elements is undefined. (14.9.36, SORT statement, general rule 4)

53) SORT statement. The results of the execution of a file SORT statement are undefined when the file specified in the USING phrase is in an open mode. (14.9.36, SORT statement, general rule 9a)

54) SORT statement. The results of the execution of a file SORT statement are undefined when the file specified in the GIVING phrase is in an open mode. (14.9.36, SORT statement, general rule 9c)

55) SORT Statement. The results of the execution of a file SORT statement are undefined if the range of the input procedure causes the execution of any MERGE, RETURN, or file SORT statement. (14.9.36, SORT statement, general rule 10)

56) SORT Statement. The results of the execution of a file SORT statement are undefined if the range of the output procedure causes the execution of any MERGE, RELEASE, or file SORT statement. (14.9.36, SORT statement, general rule 13)

57) SORT statement. For a relative file, the content of the relative key data item is undefined after the execution of the SORT statement if the USING file is not referenced in the GIVING phrase. (14.9.36, SORT statement, general rule 12c)

58) SORT statement. The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the file description entry for the USING file is undefined upon completion of the SORT statement for which the USING phrase is specified. (14.9.36, SORT statement, general rule 12)

59) SORT statement. The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the sort-merge file description entry for file SORT file name is undefined upon completion of the SORT statement for which the GIVING phrase is specified. (14.9.36, SORT statement, general rule 15)

60) START statement. Following the unsuccessful execution of the START statement for a given indexed file, the key of reference for that file is undefined. (14.9.37, START statement, general rule 7)

61) STRING statement. If the STRING identifier, or DELIMITED BY identifier, occupies the same storage area as the INTO identifier, or WITH POINTER identifier, or if the INTO identifier and the WITH POINTER identifier occupy the same storage area, the result of the execution of the STRING statement is undefined, even if they are defined by the same data description entry. (14.9.39, STRING statement, general rule 10 and 14.6.9, Overlapping operands)

62) UNSTRING statement. If the UNSTRING identifier, DELIMITED BY identifier, or OR identifier, occupies the same storage area as the INTO identifier, DELIMITER identifier, COUNT identifier, POINTER identifier, or

TALLYING identifier, or if the INTO identifier, DELIMITER identifier, or COUNT identifier, occupies the same storage area as the POINTER identifier or TALLYING identifier, or if the POINTER identifier and the TALLYING identifier occupy the same storage area, the result of the execution of the UNSTRING statement is undefined, even if they are defined by the same data description entry. (14.9.44, UNSTRING statement, general rule 18 and 14.6.9, Overlapping operands)

63) VALIDATE statement. If the evaluation of an arithmetic expression specified in the VARYING clause of an element of the operand of a VALIDATE statement produces a noninteger value, the content of the receiving items of the VALIDATE statement is undefined. (13.18.63, VARYING clause, general rule 6)

64) VARYING clause. If the value of an arithmetic expression in the VARYING clause produces a noninteger value, the content of the print line is undefined. (13.18.63, VARYING clause, general rule 5)

65) WRITE statement. The content of the bytes that extend beyond the end of the record are undefined when the number of bytes to be written to the file is greater than the number of bytes in the record. (14.9.47, WRITE statement, general rule 13)

66) WRITE statement. If two or more file connectors for a sequential file add records by sharing the physical file after opening it in extend mode, the added records follow the records present in the physical file when it was opened, but are otherwise in an undefined order. (14.9.47, WRITE statement, general rule 19)

67) WRITE statement (ADVANCING phrase). If the value of the data item referenced by the ADVANCING identifier is negative, the results of the ADVANCING operation are undefined. (14.9.47, WRITE statement, general rule 22b)

A.3 Processor-dependent language element list

A processor consists of the hardware and associated software used to translate a compilation group or to execute a run unit.

The following is a list of the COBOL language elements within this final committee draft International Standard that depend on specific devices or on a specific processor capability, functionality, or architecture. A processor-dependent element may relate to capability or functionality of hardware or of software, or both. An element described with a device-specific term may be implemented in hardware, software, or a combination of hardware and software.

1) The CURSOR and CRT STATUS clauses in the SPECIAL-NAMES paragraph, the screen section, the screen format of the ACCEPT statement, and the screen format of the DISPLAY statement are dependent on the functionality of the processor.

2) The rendition of the monochromatic, polychromatic or other attributes specified by the following clauses is dependent upon a terminal device capable of supporting them:

   BACKGROUND-COLOR
   BELL
   BLINK
   FOREGROUND-COLOR
   HIGHLIGHT
   LOWLIGHT
   REVERSE-VIDEO
   UNDERLINE

3) The ability to specify a significand longer than 31 digits in a floating-point literal is a processor-dependent element of standard COBOL.

4) The availability of the ARITHMETIC IS STANDARD, ARITHMETIC IS STANDARD-BINARY, and ARITHMETIC IS STANDARD-DECIMAL clauses of the OPTIONS paragraph is dependent on the capabilities of the processor.

5) The availability of the NEAREST-EVEN and PROHIBITED phrases of the INTERMEDIATE ROUNDING clause of the OPTIONS paragraph is dependent on the capabilities of the processor.

6) The availability of the NEAREST-EVEN, NEAREST-AWAY-FROM-ZERO, NEAREST-TOWARD-ZERO, PROHIBITED, TOWARD-GREATER and TOWARD-LESSER subphrases of the ROUNDED phrase is dependent on the capabilities of the processor.

7) The USAGE BINARY clause is dependent upon the availability of a suitable computer architecture for the binary data format.

8) The usages BINARY-CHAR, BINARY-SHORT, BINARY-LONG, and BINARY-DOUBLE are dependent upon the availability of a suitable computer architecture for binary data formats.

9) The usages FLOAT-BINARY-7, FLOAT-BINARY-16, and FLOAT-BINARY-34 are dependent on the availability of a suitable computer architecture for the basic binary floating-point formats Binary32, Binary64 and Binary128 as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.4, Binary interchange format encodings.

10) The usages FLOAT-DECIMAL-16 and FLOAT-DECIMAL-34 are dependent on the availability of a suitable computer architecture for the basic decimal floating-point formats Decimal64 and Decimal128 with decimal encoding of the significand, as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 3.5, Decimal interchange format encodings.

11) The usages FLOAT-SHORT, FLOAT-LONG, and FLOAT-EXTENDED are dependent upon the availability of a suitable computer architecture for floating-point data formats.

12) The USAGE PACKED-DECIMAL clause is dependent upon the availability of a suitable computer architecture for the packed decimal data format.

13) If positioning is not applicable on the hardware device, the operating system will ignore the positioning specified or implied by the DISPLAY statement.

14) The STANDARD-COMPARE intrinsic function, the EC-ORDER-NOT-SUPPORTED exception condition, and the ORDER TABLE clause in the SPECIAL-NAMES paragraph are dependent upon an implementation of ISO/IEC 14651. The implementor is not required to accept the syntax or to set the EC-ORDER-NOT-SUPPORTED exception condition to exist when support for ISO/IEC 14651 is not provided.

15) The STANDARD-1 phrase of the RECORD DELIMITER clause is dependent upon a reel type of device.

16) The CODE-SET clause is dependent upon a device capable of supporting the specified code.

17) The REEL/UNIT phrase of the CLOSE statement is dependent upon a reel or mass storage type of device.

18) The FOR REMOVAL phrase of the CLOSE statement is dependent upon a reel or mass storage type of device.

19) The WITH NO REWIND phrase of the CLOSE statement is dependent upon a reel or mass storage type of device.

20) The DELETE statement is dependent upon a mass storage device.

21) The I-O phrase of the OPEN statement is dependent upon a mass storage type of device.

22) The WITH NO REWIND phrase of the OPEN statement is dependent upon a reel or mass storage type of device.

23) The EXTEND phrase of the OPEN statement is dependent upon a reel or mass storage type of device.

24) The REWRITE statement is dependent upon a mass storage type of device.

25) The I-O phrase of the USE statement is dependent upon a mass storage type of device.

26) The BEFORE/AFTER ADVANCING phrase of the WRITE statement is dependent upon a device capable of vertical positioning or of an action based on mnemonic-name.

27) The capability of creating, printing, and displaying source code written with extended letters and national literals and the capability of printing or displaying data containing national characters is dependent on devices capable of processing the coded character sets supported by the implementation.

28) The SHARING clause in the file control entry and the SHARING phrase on the OPEN statement are dependent on the capabilities of the processor.

29) The PREVIOUS phrase of the READ statement and the relational operators LESS, NOT GREATER, or LESS OR EQUAL in the START statement are dependent on the capability of the operating environment to access records in a reverse order.

30) The capability of specifying the SOURCE phrase of the RECORD KEY clause is dependent on an operating environment that supports this capability.

31) The capability of specifying a collating sequence for primary and alternate keys of an indexed file where the alphabet specified in the COLLATING SEQUENCE clause is defined in the SPECIAL-NAMES paragraph with the LOCALE phrase or with literals is dependent on a processor that supports such alphabets for use with indexes.

32) The LOCK MODE clause in the file control entry and all of the record locking phrases on the input-output statements are dependent on the capability of the processor.

33) The functionality of culturally-dependent collating sequences, multiple alternate record keys with different collating sequences, and collating sequences different from the primary record key is dependent on the capabilities of the processor.

34) The capabilities of specifying a zero-length record for relative and sequential files, and of reading and writing such records, are dependent on the capabilities of the processor.

35) The capability of indicating abnormal termination of the run unit is dependent on the capabilities of the processor.

36) When parametric polymorphism is provided in object orientation, the algorithm used for method resolution may be dependent on the methodology used in the object management system.

37) The ability to detect any specific level-3 exception condition is dependent on the ability of the processor to detect that level-3 exception condition.

A.4 Optional language element list

The following is a list of the language elements that are optional language elements within this final committee draft International Standard. An implementation is required to accept the syntax and provide the functionality for an optional element only when support for that language element is claimed by the implementor.

This list identifies features, statements, formats (or parts of formats), clauses, or phrases that are optional. Any associated syntax rules, general rules, other rules, exception conditions, and I-O status values are also optional, even if not explicitly listed.

NOTE 1 For some optional elements, higher-level language constructs are shown to provide context. Do not construe those higher-level constructs or cross-referenced topics as also optional. For example, only the lowest-level element (repetition of interface-name-2) in the following entry is optional:

CLASS-ID paragraph, INHERITS phrase, repetition of interface-name-2 (11.2)

An implementor's user documentation shall identify optional language elements for which support is claimed.

The requirements of A.1, Implementor-defined language element list, apply for all supported optional language elements.

A.4.1 ACCEPT and DISPLAY screen handling

1) Screens (9.2)
2) SPECIAL-NAMES paragraph, CURSOR clause and CRT-STATUS clause (12.3.6)
3) Data division, SCREEN SECTION header (13.9)
4) Screen section (13.9)
5) Screen description entry, format 1: group; format 2: elementary (13.17)
6) AUTO clause (13.18.3)
7) BACKGROUND-COLOR clause (13.18.4)
8) BELL clause (13.18.6)
9) BLANK clause (13.18.7)
10) BLINK clause (13.18.9)
11) COLUMN clause, format 2: screen-item (13.18.14)
12) Entry-name clause, format 2: screen-name (13.18.19)
13) ERASE clause (13.18.20)
14) FOREGROUND-COLOR clause (13.18.22)
15) FROM clause (13.18.24)
16) FULL clause (13.18.25)
17) HIGHLIGHT clause (13.18.29)
18) LOWLIGHT clause (13.18.35)
19) REVERSE-VIDEO clause (13.18.47)
20) REQUIRED clause (13.18.46)
21) TO clause (13.18.55)
22) UNDERLINE clause (13.18.58)
23) ACCEPT statement, format 3: screen (14.9.1)
24) DISPLAY statement, format 2: screen (14.9.9)
25) EC-SCREEN exception conditions in the RAISING phrase of the EXIT and GOBACK statements, the RAISING phrase of the procedure division header, the USE statement, the RAISE statement, and the TURN compiler directive.

A.4.2 ANY LENGTH data items

1) ANY LENGTH clause (13.18.2)

A.4.3 ARITHMETIC IS STANDARD

1) OPTIONS paragraph, STANDARD phrase of the ARITHMETIC clause (11.9.4)

A.4.4 COBOL character repertoire

1) Extended letters in source code (8.1.2)

A.4.5 Dynamic capacity tables

1) OCCURS clause, format 4: dynamic-capacity-table (13.18.37)

2) EC-BOUND-OVERFLOW, EC-BOUND-SET, EC-BOUND-TABLE-LIMIT, and EC-FLOW-SEARCH exception conditions in the RAISING phrase of the EXIT and GOBACK statements, the RAISING phrase of the procedure division header, the USE statement, the RAISE statement, and the TURN compiler directive

A.4.6 Exception handling

1) RESUME statement (14.9.31)

A.4.7 File sharing and record locking

1) Sharing mode (9.1.14)

2) Record locking (9.1.15)

3) File control entry, LOCK MODE clause and SHARING clause (12.4.3)

4) LOCK MODE clause (12.4.4.8)

5) SHARING clause (12.4.4.14)

6) OPEN statement, SHARING phrase (14.9.26)

7) EC-I-O-FILE-SHARING exception condition in the RAISING phrase of the EXIT and GOBACK statements, the RAISING phrase of the procedure division header, the USE statement, the RAISE statement, and the TURN compiler directive

A.4.8 FORMAT and SELECT WHEN file handling

1) FORMAT clause (13.18.23

2) SELECT WHEN clause (13.18.50}

A.4.9 Language fundamentals

1) Extended letters in the COBOL character repertoire (8.1.2)

A.4.10 Locale support and related functions

1) OBJECT-COMPUTER paragraph, CHARACTER CLASSIFICATION clause (12.3.5)

2) SPECIAL-NAMES paragraph: LOCALE clause and LOCALE phrases in the ALPHABET clause (12.3.6)

3) LOCALE-COMPARE function (15.47)

4) LOCALE-DATE function (15.48)

5) LOCALE-TIME function (15.49)

6) LOCALE-TIME-FROM-SECONDS function (15.50)

7) LOWER-CASE function, LOCALE keyword (15.53)

8) UPPER-CASE function, LOCALE keyword (15.90)

9) STANDARD-COMPARE function (15.79)

10) TEST-NUMVAL-C function, LOCALE keyword and locale-name-1 (15.87)

11) PICTURE clause, format 2: locale (13.18.39)

12) SET statement, format 11: set-locale and format 12: save-locale (14.9.35)

13) EC-LOCALE and EC-ORDER-NOT-SUPPORTED exception conditions in the RAISING phrase of the EXIT and GOBACK statements, the RAISING phrase of the procedure division header, the USE statement, the RAISE statement, and the TURN compiler directive

A.4.11 Object orientation

1) Multiple inheritance: CLASS-ID paragraph, INHERITS phrase, repetition of class-name-2 (11.3)

2) IMultiple inheritance: NTERFACE-ID paragraph, INHERITS phrase, repetition of interface-name-2 (11.6)

3) Parametric polymorphism (9.3.5.2)

A.4.12 Report Writer

1) Data division, REPORT SECTION header (13)

2) File description entry, format 3: report (13.4.4)

3) Report section (13.8)

4) Report description entry (13.8.3)

5) Report group description entry (13.8.4)

6) CODE clause (13.18.12)

7) COLUMN clause, format 1: report-writer (13.18.14)

8) CONTROL clause (13.18.16)

9) GROUP INDICATE clause (13.18.27)

10) LINAGE clause (13.18.33)

11) NEXT GROUP clause (13.18.36)

12) OCCURS clause, format 3: report-writer (13.18.37)

13) PAGE clause (13.18.38)

14) PRESENT WHEN clause (13.18.40)

15) REPORT clause (13.18.45)

16) SOURCE clause (13.18.52)

17) SUM clause (13.18.53)

18) TYPE clause, report-group format (13.18.56)

19) VALUE clause, format 4: report-section format (13.18.62)

20) VARYING clause (13.18.63)

21) GENERATE statement (14.9.15)

22) INITIATE statement (14.9.20)

23) SUPPRESS statement (14.9.41)

24) TERMINATE statement (14.9.42)

25) EC-REPORT and EC-FLOW-REPORT exception conditions in the RAISING phrase of the EXIT and GOBACK statements, the RAISING phrase of the procedure division header, the USE statement, the RAISE statement, and the TURN compiler directive

A.4.13 REWRITE FILE and WRITE FILE

1) REWRITE FILE file-name-1 (14.9.33)

2) WRITE FILE file-name-1 (14.9.47)

A.4.14 VALIDATE

1) Data description entry, format 4: validation (13.16)

2) DEFAULT clause (13.18.17)

3) DESTINATION clause (13.18.18)

4) INVALID clause (13.18.30)

5) PRESENT WHEN clause (13.18.40)

6) VALIDATE-STATUS clause (13.18.61)

7) VALUE clause, format 4: content-validation-entry (13.18.62)

8) VARYING clause (13.18.63)

9) VALIDATE statement (13.18.63

10) EC-VALIDATE exception conditions in the RAISING phrase of the EXIT and GOBACK statements, the RAISING phrase of the procedure division header, the USE statement, the RAISE statement, and the TURN compiler directive

Annex B
(normative)

Characters permitted in user-defined words

B.1 General

This annex presents the repertoire of characters permitted in user-defined words. The characters include the letters, ideographic and syllabic characters, digits, modifiers, and combining marks recommended for programming language identifiers in Annex A of ISO/IEC TR 10176:2003. These characters can be used to write many natural languages of the world.

This final committee draft International Standard adds three characters beyond those specified in ISO/IEC TR 10176:2003, Middle Dot (00B7), which is used in writing Catalan, hyphen-minus, and underscore.

The repertoire includes the basic letters, basic digits, and basic hyphen and underscore; these are the basic characters of the COBOL character repertoire that are permitted in user-defined words. The remaining characters of the repertoire are referred to as extended letters, which are permitted in user-defined words as specified in the detailed rules of COBOL. The basic letters and extended letters can be used to write many natural languages of the world.

NOTE Attention is drawn to the possibility that use of extended letters in user-defined words can impact source code portability because the full set of extended letters is not universally implemented at the time of writing this final committee draft International Standard.

B.2 Notation

The repertoire of characters permitted in user-defined words is defined using the hexadecimal code values that identify letters specified in ISO/IEC 10646-1. A range of characters is specified in the form nnnn-mmmm, where nnnn identifies the first character in the range and mmmm identifies the last character in the range.

NOTE 1 The specification of code values in ISO/IEC 10646 is a means of character identification and not a requirement for implementation of ISO/IEC 10646.

NOTE 2 The letter 'U' customarily used in referencing code values in ISO/IEC 10646 is not used so as to increase the readability of code values and range specifications.

B.3 Repertoire of characters permitted in user-defined words

B.3.1 Characters from TR 10176:2003, Annex A

The following list presents the characters identified in Annex A of TR 10176:2003 as recommended for use in programming language identifiers:

NOTE This repertoire includes combining marks specified for level 2 of ISO/IEC 10646. Combining marks of level 3 are excluded so as to avoid alternative representations of user-defined words.

Script or character classification	**Letters or range of letters**
Latin:	0041-005A, 0061-007A, 00AA, 00BA, 00C0-00D6, 00D8-00F6, 00F8-01BA, 01BB, 01BC-01BF, 01C0-01C3, 01C4-021F, 0222-0233, 0250-02AD, 1E00-1E9B, 1EA0-1EF9, 207F

Greek:	0386, 0388-038A, 038C, 038E-03A1, 03A3-03CE, 03D0-03D7, 03DA-03F3, 1F00-1F15, 1F18-1F1D, 1F20-1F45, 1F48-1F4D, 1F50-1F57, 1F59, 1F5B, 1F5D, 1F5F-1F7D, 1F80-1FB4, 1FB6-1FBC, 1FC2-1FC4, 1FC6-1FCC, 1FD0-1FD3, 1FD6-1FDB, 1FE0-1FEC, 1FF2-1FF4, 1FF6-1FFC
Cyrillic:	0400-0481, 048C-04C4, 04C7-04C8, 04CB-04CC, 04D0-04F5, 04F8-04F9
Armenian:	0531-0556, 0561-0587
Hebrew:	05D0-05EA, 05F0-05F2
Hebrew (combining):	05B0-05B9, 05BB-05BD, 05BF, 05C1-05C2
Arabic:	0621-063A, 0640, 0641-064A, 0671-06D3, 06D5, 06E5-06E6, 06FA-06FC
Arabic: (Combining	064B-0652, 0670, 06D6-06DC, 06E7-06E8, 06EA-06ED
Syriac:	0710, 0712-072C
Syriac: (Combining)	0711
Thaana:	0780-07A5
Thaana: (combining)	07A6-07B0
Devanagari:	0905-0939, 093D, 0950, 0958-0961
Devanagari: (combining)	0901-0902, 0903, 093E-0940, 0941-0948, 0949-094C, 094D, 0950, 0951-0952, 0962-0963
Bengali:	985-098C, 098F-0990, 0993-09A8, 09AA-09B0, 09B2, 09B6-09B9, 09DC-09DD, 09DF-09E1, 09F0-09F1
Bengali: (combining)	0981, 0982-0983, 09BE-09C0, 09C1-09C4, 09C7-09C8, 09CB-09CC, 09CD, 09E2-09E3
Gurmukhi:	A05-0A0A, 0A0F-0A10, 0A13-0A28, 0A2A-0A30, 0A32-0A33, 0A35-0A36, 0A38-0A39, 0A59-0A5C, 0A5E, 0A72-0A74
Gurmukhi: (combining)	0A02, 0A3E-0A40, 0A41-0A42, 0A47-0A48, 0A4B-0A4D
Gujarati:	0A85-0A8B, 0A8D, 0A8F-0A91, 0A93-0AA8, 0AAA-0AB0, 0AB2-0AB3, 0AB5-0AB9, 0ABD, 0AD0, 0AE0
Gujarati: (combining)	0A81-0A82, 0A83, 0ABE-0AC0, 0AC1-0AC5, 0AC7-0AC8, 0AC9, 0ACB-0ACC, 0ACD
Oriya:	0B05-0B0C, 0B0F-0B10, 0B13-0B28, 0B2A-0B30, 0B32-0B33, 0B36-0B39, 0B3D, 0B5C-0B5D, 0B5F-0B61
Oriya: (combining)	0B01, 0B02-0B03, 0B3E, 0B3F, 0B40, 0B41-0B43, 0B47-0B48, 0B4B-0B4C, 0B4D

Tamil:	0B85-0B8A, 0B8E-0B90, 0B92-0B95, 0B99-0B9A, 0B9C, 0B9E-0B9F, 0BA3-0BA4, 0BA8-0BAA, 0BAE-0BB5, 0BB7-0BB9
Tamil: (combining)	0B82, 0B83, 0BBE-0BBF, 0BC0, 0BC1-0BC2, 0BC6-0BC8, 0BCA-0BCC, 0BCD
Telugu:	0C05-0C0C, 0C0E-0C10, 0C12-0C28, 0C2A-0C33, 0C35-0C39, 0C60-0C61
Telugu: (combining)	0C01-0C03, 0C3E-0C40, 0C41-0C44, 0C46-0C48, 0C4A-0C4D
Kannada:	0C85-0C8C, 0C8E-0C90, 0C92-0CA8, 0CAA-0CB3, 0CB5-0CB9, 0CDE, 0CE0-0CE1
Kannada: (combining)	0C82-0C83, 0CBE, 0CBF, 0CC0-0CC4, 0CC6, 0CC7-0CC8, 0CCA-0CCB, 0CCC-0CCD
Malayalam:	0D05-0D0C, 0D0E-0D10, 0D12-0D28, 0D2A-0D39, 0D60-0D61
Malayalam: (combining)	0D02-0D03, 0D3E-0D40, 0D41-0D43, 0D46-0D48, 0D4A-0D4C, 0D4D
Sinhala:	0D85-0D96, 0D9A-0DB1, 0DB3-0DBB, 0DBD, 0DC0-0DC6
Sinhala: (combining)	0D82-0D83, 0DCA, 0DCF-0DD1, 0DD2-0DD4, 0DD6, 0DD8-0DDF, 0DF2-0DF3
Thai:	0E01-0E30, 0E32-0E33, 0E40-0E45, 0E46
Thai: (combining)	0E31, 0E34-0E3A, 0E47-0E4E
Lao:	0E81-0E82, 0E84, 0E87-0E88, 0E8A, 0E8D, 0E94-0E97, 0E99-0E9F, 0EA1-0EA3, 0EA5, 0EA7, 0EAA-0EAB, 0EAD-0EB0, 0EB2-0EB3, 0EBD, 0EC0-0EC4, 0EC6, 0EDC-0EDD
Lao: (combining)	0EB1, 0EB4-0EB9, 0EBB-0EBC, 0EC8-0ECD
Tibetan:	0F00, 0F40-0F47, 0F49-0F6A, 0F88-0F8B
Tibetan: (combining)	0F18-0F19, 0F35, 0F37, 0F39, 0F71-0F7E, 0F7F, 0F80-0F84, 0F86-0F87, 0F90-0F97, 0F99-0FBC
Myanmar:	1000-1021, 1023-1027, 1029-102A, 1050-1055
Myanmar: (combining)	102C, 102D-1030, 1031, 1032, 1036-1037, 1038, 1039, 1056-1057, 1058-1059
Georgian:	10A0-10C5, 10D0-10F6
Ethiopic:	1200-1206, 1208-1246, 1248, 124A-124D, 1250-1256, 1258, 125A-125D, 1260-1286, 1288, 128A-128D, 1290-12AE, 12B0, 12B2-12B5, 12B8-12BE, 12C0, 12C2-12C5, 12C8-12CE, 12D0-12D6, 12D8-12EE, 12F0-130E, 1310, 1312-1315, 1318-131E, 1320-1346, 1348-135A
Cherokee:	13A0-13F4
Canadian Aboriginal Syllabics :	1401-166C, 166F-1676

Ogham:	1681-169A
Runic:	16A0-16EA, 16EE-16F0
Khmer:	1780-17B3
Khmer: (combining)	17B4-17B6, 17B7-17BD, 17BE-17C5, 17C6, 17C7-17C8, 17C9-17D3
Mongolian:	1820-1842, 1843, 1844-1877, 1880-18A8
Mongolian: (combining)	18A9
Hiragana:	3041-3094
Katakana:	30A1-30FA, 30FB, 30FC
Bopomofo:	3105-312C, 31A0-31B7
CJK Unified Ideographs:	3400-4DB5, 4E00-9FA5, FA0E-FA0F, FA11, FA13-FA14, FA1F, FA21, FA23-FA24, FA27-FA29
Yi:	A000-A48C
Hangul:	AC00-D7A3
Digits:	0030-0039, 0660-0669, 06F0-06F9, 0966-096F, 09E6-09EF, 0A66-0A6F, 0AE6-0AEF, 0B66-0B6F, 0BE7-0BEF, 0C66-0C6F, 0CE6-0CEF, 0D66-0D6F, 0E50-0E59, 0ED0-0ED9, 0F20-0F29, 1040-1049, 1369-1371, 17E0-17E9, 1810-1819
Special characters:	00B5, 02B0-02B8, 02BB-02C1, 02D0-02D1, 02E0-02E4, 02EE, 037A, 0559, 1FBE, 203F-2040, 2102, 2107, 210A-2113, 2115, 2119-211D, 2124, 2126, 2128, 212A-212D, 212F-2131, 2133-2134, 2135-2138, 2139, 2160-2183, 3005, 3006, 3007, 3021-3029, 3038-303A

B.3.2 Additional characters

Special characters:

002D (hyphen-minus)
003F (underscore, low line)
00B7 (middle dot)

Annex C
(normative)

Mapping of uppercase letters to lowercase letters

This annex shows the mapping of uppercase letters to lowercase letters in the COBOL character repertoire.

C.1 Notations

The following notation conventions are used in this annex:

1) Ummmm, where mmmm is a four-digit number in hexadecimal notation using the digits 0-9 and the letters A-F (for 10 through 15, respectively) represents a code value that identifies a letter specified in ISO/IEC 10646.

   NOTE The specification of code values is a means of character identification and not a requirement for implementation of ISO/IEC 10646.

2) (<Uxxxx>,<Uyyyy>) denotes a mapping of an uppercase letter Uxxxx to a lowercase letter Uyyyy.

C.2 General case mappings

The mapping of uppercase letters to lowercase letters is listed below.

NOTE Each character listed in Annex B, Characters permitted in user-defined words, was checked against UnicodeData.txt for Unicode 5.2.0 and if a corresponding lowercase letter was specified, the character is listed with its corresponding lowercase letter below.

```
(<U0041>,<U0061>);(<U0042>,<U0062>);(<U0043>,<U0063>);(<U0044>,<U0064>);
(<U0045>,<U0065>);(<U0046>,<U0066>);(<U0047>,<U0067>);(<U0048>,<U0068>);
(<U0049>,<U0069>);(<U004A>,<U006A>);(<U004B>,<U006B>);(<U004C>,<U006C>);
(<U004D>,<U006D>);(<U004E>,<U006E>);(<U004F>,<U006F>);(<U0050>,<U0070>);
(<U0051>,<U0071>);(<U0052>,<U0072>);(<U0053>,<U0073>);(<U0054>,<U0074>);
(<U0055>,<U0075>);(<U0056>,<U0076>);(<U0057>,<U0077>);(<U0058>,<U0078>);
(<U0059>,<U0079>);(<U005A>,<U007A>);(<U00C0>,<U00E0>);(<U00C1>,<U00E1>);
(<U00C2>,<U00E2>);(<U00C3>,<U00E3>);(<U00C4>,<U00E4>);(<U00C5>,<U00E5>);
(<U00C6>,<U00E6>);(<U00C7>,<U00E7>);(<U00C8>,<U00E8>);(<U00C9>,<U00E9>);
(<U00CA>,<U00EA>);(<U00CB>,<U00EB>);(<U00CC>,<U00EC>);(<U00CD>,<U00ED>);
(<U00CE>,<U00EE>);(<U00CF>,<U00EF>);(<U00D0>,<U00F0>);(<U00D1>,<U00F1>);
(<U00D2>,<U00F2>);(<U00D3>,<U00F3>);(<U00D4>,<U00F4>);(<U00D5>,<U00F5>);
(<U00D6>,<U00F6>);(<U00D8>,<U00F8>);(<U00D9>,<U00F9>);(<U00DA>,<U00FA>);
(<U00DB>,<U00FB>);(<U00DC>,<U00FC>);(<U00DD>,<U00FD>);(<U00DE>,<U00FE>);
(<U0100>,<U0101>);(<U0102>,<U0103>);(<U0104>,<U0105>);(<U0106>,<U0107>);
(<U0108>,<U0109>);(<U010A>,<U010B>);(<U010C>,<U010D>);(<U010E>,<U010F>);
(<U0110>,<U0111>);(<U0112>,<U0113>);(<U0114>,<U0115>);(<U0116>,<U0117>);
(<U0118>,<U0119>);(<U011A>,<U011B>);(<U011C>,<U011D>);(<U011E>,<U011F>);
(<U0120>,<U0121>);(<U0122>,<U0123>);(<U0124>,<U0125>);(<U0126>,<U0127>);
(<U0128>,<U0129>);(<U012A>,<U012B>);(<U012C>,<U012D>);(<U012E>,<U012F>);
(<U0130>,<U0069>);(<U0132>,<U0133>);(<U0134>,<U0135>);(<U0136>,<U0137>);
(<U0139>,<U013A>);(<U013B>,<U013C>);(<U013D>,<U013E>);(<U013F>,<U0140>);
(<U0141>,<U0142>);(<U0143>,<U0144>);(<U0145>,<U0146>);(<U0147>,<U0148>);
(<U014A>,<U014B>);(<U014C>,<U014D>);(<U014E>,<U014F>);(<U0150>,<U0151>);
(<U0152>,<U0153>);(<U0154>,<U0155>);(<U0156>,<U0157>);(<U0158>,<U0159>);
```

```
(<U015A>,<U015B>);(<U015C>,<U015D>);(<U015E>,<U015F>);(<U0160>,<U0161>);
(<U0162>,<U0163>);(<U0164>,<U0165>);(<U0166>,<U0167>);(<U0168>,<U0169>);
(<U016A>,<U016B>);(<U016C>,<U016D>);(<U016E>,<U016F>);(<U0170>,<U0171>);
(<U0172>,<U0173>);(<U0174>,<U0175>);(<U0176>,<U0177>);(<U0178>,<U00FF>);
(<U0179>,<U017A>);(<U017B>,<U017C>);(<U017D>,<U017E>);(<U0181>,<U0253>);
(<U0182>,<U0183>);(<U0184>,<U0185>);(<U0186>,<U0254>);(<U0187>,<U0188>);
(<U0189>,<U0256>);(<U018A>,<U0257>);(<U018B>,<U018C>);(<U018E>,<U01DD>);
(<U018F>,<U0259>);(<U0190>,<U025B>);(<U0191>,<U0192>);(<U0193>,<U0260>);
(<U0194>,<U0263>);(<U0196>,<U0269>);(<U0197>,<U0268>);(<U0198>,<U0199>);
(<U019C>,<U026F>);(<U019D>,<U0272>);(<U019F>,<U0275>);(<U01A0>,<U01A1>);
(<U01A2>,<U01A3>);(<U01A4>,<U01A5>);(<U01A6>,<U0280>);(<U01A7>,<U01A8>);
(<U01A9>,<U0283>);(<U01AC>,<U01AD>);(<U01AE>,<U0288>);(<U01AF>,<U01B0>);
(<U01B1>,<U028A>);(<U01B2>,<U028B>);(<U01B3>,<U01B4>);(<U01B5>,<U01B6>);
(<U01B7>,<U0292>);(<U01B8>,<U01B9>);(<U01BC>,<U01BD>);(<U01C4>,<U01C6>);
(<U01C5>,<U01C6>);(<U01C7>,<U01C9>);(<U01C8>,<U01C9>);(<U01CA>,<U01CC>);
(<U01CB>,<U01CC>);(<U01CD>,<U01CE>);(<U01CF>,<U01D0>);(<U01D1>,<U01D2>);
(<U01D3>,<U01D4>);(<U01D5>,<U01D6>);(<U01D7>,<U01D8>);(<U01D9>,<U01DA>);
(<U01DB>,<U01DC>);(<U01DE>,<U01DF>);(<U01E0>,<U01E1>);(<U01E2>,<U01E3>);
(<U01E4>,<U01E5>);(<U01E6>,<U01E7>);(<U01E8>,<U01E9>);(<U01EA>,<U01EB>);
(<U01EC>,<U01ED>);(<U01EE>,<U01EF>);(<U01F1>,<U01F3>);(<U01F2>,<U01F3>);
(<U01F4>,<U01F5>);(<U01F6>,<U0195>);(<U01F7>,<U01BF>);(<U01F8>,<U01F9>);
(<U01FA>,<U01FB>);(<U01FC>,<U01FD>);(<U01FE>,<U01FF>);(<U0200>,<U0201>);
(<U0202>,<U0203>);(<U0204>,<U0205>);(<U0206>,<U0207>);(<U0208>,<U0209>);
(<U020A>,<U020B>);(<U020C>,<U020D>);(<U020E>,<U020F>);(<U0210>,<U0211>);
(<U0212>,<U0213>);(<U0214>,<U0215>);(<U0216>,<U0217>);(<U0218>,<U0219>);
(<U021A>,<U021B>);(<U021C>,<U021D>);(<U021E>,<U021F>);(<U0222>,<U0223>);
(<U0224>,<U0225>);(<U0226>,<U0227>);(<U0228>,<U0229>);(<U022A>,<U022B>);
(<U022C>,<U022D>);(<U022E>,<U022F>);(<U0230>,<U0231>);(<U0232>,<U0233>);
(<U0386>,<U03AC>);(<U0388>,<U03AD>);(<U0389>,<U03AE>);(<U038A>,<U03AF>);
(<U038C>,<U03CC>);(<U038E>,<U03CD>);(<U038F>,<U03CE>);(<U0391>,<U03B1>);
(<U0392>,<U03B2>);(<U0393>,<U03B3>);(<U0394>,<U03B4>);(<U0395>,<U03B5>);
(<U0396>,<U03B6>);(<U0397>,<U03B7>);(<U0398>,<U03B8>);(<U0399>,<U03B9>);
(<U039A>,<U03BA>);(<U039B>,<U03BB>);(<U039C>,<U03BC>);(<U039D>,<U03BD>);
(<U039E>,<U03BE>);(<U039F>,<U03BF>);(<U03A0>,<U03C0>);(<U03A1>,<U03C1>);
(<U03A3>,<U03C3>);(<U03A4>,<U03C4>);(<U03A5>,<U03C5>);(<U03A6>,<U03C6>);
(<U03A7>,<U03C7>);(<U03A8>,<U03C8>);(<U03A9>,<U03C9>);(<U03AA>,<U03CA>);
(<U03AB>,<U03CB>);(<U03DA>,<U03DB>);(<U03DC>,<U03DD>);(<U03DE>,<U03DF>);
(<U03E0>,<U03E1>);(<U03E2>,<U03E3>);(<U03E4>,<U03E5>);(<U03E6>,<U03E7>);
(<U03E8>,<U03E9>);(<U03EA>,<U03EB>);(<U03EC>,<U03ED>);(<U03EE>,<U03EF>);
(<U0400>,<U0450>);(<U0401>,<U0451>);(<U0402>,<U0452>);(<U0403>,<U0453>);
(<U0404>,<U0454>);(<U0405>,<U0455>);(<U0406>,<U0456>);(<U0407>,<U0457>);
(<U0408>,<U0458>);(<U0409>,<U0459>);(<U040A>,<U045A>);(<U040B>,<U045B>);
(<U040C>,<U045C>);(<U040D>,<U045D>);(<U040E>,<U045E>);(<U040F>,<U045F>);
(<U0410>,<U0430>);(<U0411>,<U0431>);(<U0412>,<U0432>);(<U0413>,<U0433>);
(<U0414>,<U0434>);(<U0415>,<U0435>);(<U0416>,<U0436>);(<U0417>,<U0437>);
(<U0418>,<U0438>);(<U0419>,<U0439>);(<U041A>,<U043A>);(<U041B>,<U043B>);
(<U041C>,<U043C>);(<U041D>,<U043D>);(<U041E>,<U043E>);(<U041F>,<U043F>);
(<U0420>,<U0440>);(<U0421>,<U0441>);(<U0422>,<U0442>);(<U0423>,<U0443>);
(<U0424>,<U0444>);(<U0425>,<U0445>);(<U0426>,<U0446>);(<U0427>,<U0447>);
(<U0428>,<U0448>);(<U0429>,<U0449>);(<U042A>,<U044A>);(<U042B>,<U044B>);
(<U042C>,<U044C>);(<U042D>,<U044D>);(<U042E>,<U044E>);(<U042F>,<U044F>);
(<U0460>,<U0461>);(<U0462>,<U0463>);(<U0464>,<U0465>);(<U0466>,<U0467>);
(<U0468>,<U0469>);(<U046A>,<U046B>);(<U046C>,<U046D>);(<U046E>,<U046F>);
```

```
(<U0470>,<U0471>);(<U0472>,<U0473>);(<U0474>,<U0475>);(<U0476>,<U0477>);
(<U0478>,<U0479>);(<U047A>,<U047B>);(<U047C>,<U047D>);(<U047E>,<U047F>);
(<U0480>,<U0481>);(<U048C>,<U048D>);(<U048E>,<U048F>);(<U0490>,<U0491>);
(<U0492>,<U0493>);(<U0494>,<U0495>);(<U0496>,<U0497>);(<U0498>,<U0499>);
(<U049A>,<U049B>);(<U049C>,<U049D>);(<U049E>,<U049F>);(<U04A0>,<U04A1>);
(<U04A2>,<U04A3>);(<U04A4>,<U04A5>);(<U04A6>,<U04A7>);(<U04A8>,<U04A9>);
(<U04AA>,<U04AB>);(<U04AC>,<U04AD>);(<U04AE>,<U04AF>);(<U04B0>,<U04B1>);
(<U04B2>,<U04B3>);(<U04B4>,<U04B5>);(<U04B6>,<U04B7>);(<U04B8>,<U04B9>);
(<U04BA>,<U04BB>);(<U04BC>,<U04BD>);(<U04BE>,<U04BF>);(<U04C1>,<U04C2>);
(<U04C3>,<U04C4>);(<U04C7>,<U04C8>);(<U04CB>,<U04CC>);(<U04D0>,<U04D1>);
(<U04D2>,<U04D3>);(<U04D4>,<U04D5>);(<U04D6>,<U04D7>);(<U04D8>,<U04D9>);
(<U04DA>,<U04DB>);(<U04DC>,<U04DD>);(<U04DE>,<U04DF>);(<U04E0>,<U04E1>);
(<U04E2>,<U04E3>);(<U04E4>,<U04E5>);(<U04E6>,<U04E7>);(<U04E8>,<U04E9>);
(<U04EA>,<U04EB>);(<U04EC>,<U04ED>);(<U04EE>,<U04EF>);(<U04F0>,<U04F1>);
(<U04F2>,<U04F3>);(<U04F4>,<U04F5>);(<U04F8>,<U04F9>);(<U0531>,<U0561>);
(<U0532>,<U0562>);(<U0533>,<U0563>);(<U0534>,<U0564>);(<U0535>,<U0565>);
(<U0536>,<U0566>);(<U0537>,<U0567>);(<U0538>,<U0568>);(<U0539>,<U0569>);
(<U053A>,<U056A>);(<U053B>,<U056B>);(<U053C>,<U056C>);(<U053D>,<U056D>);
(<U053E>,<U056E>);(<U053F>,<U056F>);(<U0540>,<U0570>);(<U0541>,<U0571>);
(<U0542>,<U0572>);(<U0543>,<U0573>);(<U0544>,<U0574>);(<U0545>,<U0575>);
(<U0546>,<U0576>);(<U0547>,<U0577>);(<U0548>,<U0578>);(<U0549>,<U0579>);
(<U054A>,<U057A>);(<U054B>,<U057B>);(<U054C>,<U057C>);(<U054D>,<U057D>);
(<U054E>,<U057E>);(<U054F>,<U057F>);(<U0550>,<U0580>);(<U0551>,<U0581>);
(<U0552>,<U0582>);(<U0553>,<U0583>);(<U0554>,<U0584>);(<U0555>,<U0585>);
(<U0556>,<U0586>);(<U1E00>,<U1E01>);(<U1E02>,<U1E03>);(<U1E04>,<U1E05>);
(<U1E06>,<U1E07>);(<U1E08>,<U1E09>);(<U1E0A>,<U1E0B>);(<U1E0C>,<U1E0D>);
(<U1E0E>,<U1E0F>);(<U1E10>,<U1E11>);(<U1E12>,<U1E13>);(<U1E14>,<U1E15>);
(<U1E16>,<U1E17>);(<U1E18>,<U1E19>);(<U1E1A>,<U1E1B>);(<U1E1C>,<U1E1D>);
(<U1E1E>,<U1E1F>);(<U1E20>,<U1E21>);(<U1E22>,<U1E23>);(<U1E24>,<U1E25>);
(<U1E26>,<U1E27>);(<U1E28>,<U1E29>);(<U1E2A>,<U1E2B>);(<U1E2C>,<U1E2D>);
(<U1E2E>,<U1E2F>);(<U1E30>,<U1E31>);(<U1E32>,<U1E33>);(<U1E34>,<U1E35>);
(<U1E36>,<U1E37>);(<U1E38>,<U1E39>);(<U1E3A>,<U1E3B>);(<U1E3C>,<U1E3D>);
(<U1E3E>,<U1E3F>);(<U1E40>,<U1E41>);(<U1E42>,<U1E43>);(<U1E44>,<U1E45>);
(<U1E46>,<U1E47>);(<U1E48>,<U1E49>);(<U1E4A>,<U1E4B>);(<U1E4C>,<U1E4D>);
(<U1E4E>,<U1E4F>);(<U1E50>,<U1E51>);(<U1E52>,<U1E53>);(<U1E54>,<U1E55>);
(<U1E56>,<U1E57>);(<U1E58>,<U1E59>);(<U1E5A>,<U1E5B>);(<U1E5C>,<U1E5D>);
(<U1E5E>,<U1E5F>);(<U1E60>,<U1E61>);(<U1E62>,<U1E63>);(<U1E64>,<U1E65>);
(<U1E66>,<U1E67>);(<U1E68>,<U1E69>);(<U1E6A>,<U1E6B>);(<U1E6C>,<U1E6D>);
(<U1E6E>,<U1E6F>);(<U1E70>,<U1E71>);(<U1E72>,<U1E73>);(<U1E74>,<U1E75>);
(<U1E76>,<U1E77>);(<U1E78>,<U1E79>);(<U1E7A>,<U1E7B>);(<U1E7C>,<U1E7D>);
(<U1E7E>,<U1E7F>);(<U1E80>,<U1E81>);(<U1E82>,<U1E83>);(<U1E84>,<U1E85>);
(<U1E86>,<U1E87>);(<U1E88>,<U1E89>);(<U1E8A>,<U1E8B>);(<U1E8C>,<U1E8D>);
(<U1E8E>,<U1E8F>);(<U1E90>,<U1E91>);(<U1E92>,<U1E93>);(<U1E94>,<U1E95>);
(<U1EA0>,<U1EA1>);(<U1EA2>,<U1EA3>);(<U1EA4>,<U1EA5>);(<U1EA6>,<U1EA7>);
(<U1EA8>,<U1EA9>);(<U1EAA>,<U1EAB>);(<U1EAC>,<U1EAD>);(<U1EAE>,<U1EAF>);
(<U1EB0>,<U1EB1>);(<U1EB2>,<U1EB3>);(<U1EB4>,<U1EB5>);(<U1EB6>,<U1EB7>);
(<U1EB8>,<U1EB9>);(<U1EBA>,<U1EBB>);(<U1EBC>,<U1EBD>);(<U1EBE>,<U1EBF>);
(<U1EC0>,<U1EC1>);(<U1EC2>,<U1EC3>);(<U1EC4>,<U1EC5>);(<U1EC6>,<U1EC7>);
(<U1EC8>,<U1EC9>);(<U1ECA>,<U1ECB>);(<U1ECC>,<U1ECD>);(<U1ECE>,<U1ECF>);
(<U1ED0>,<U1ED1>);(<U1ED2>,<U1ED3>);(<U1ED4>,<U1ED5>);(<U1ED6>,<U1ED7>);
(<U1ED8>,<U1ED9>);(<U1EDA>,<U1EDB>);(<U1EDC>,<U1EDD>);(<U1EDE>,<U1EDF>);
(<U1EE0>,<U1EE1>);(<U1EE2>,<U1EE3>);(<U1EE4>,<U1EE5>);(<U1EE6>,<U1EE7>);
(<U1EE8>,<U1EE9>);(<U1EEA>,<U1EEB>);(<U1EEC>,<U1EED>);(<U1EEE>,<U1EEF>);
```

```
(<U1EF0>,<U1EF1>);(<U1EF2>,<U1EF3>);(<U1EF4>,<U1EF5>);(<U1EF6>,<U1EF7>);
(<U1EF8>,<U1EF9>);(<U1F08>,<U1F00>);(<U1F09>,<U1F01>);(<U1F0A>,<U1F02>);
(<U1F0B>,<U1F03>);(<U1F0C>,<U1F04>);(<U1F0D>,<U1F05>);(<U1F0E>,<U1F06>);
(<U1F0F>,<U1F07>);(<U1F18>,<U1F10>);(<U1F19>,<U1F11>);(<U1F1A>,<U1F12>);
(<U1F1B>,<U1F13>);(<U1F1C>,<U1F14>);(<U1F1D>,<U1F15>);(<U1F28>,<U1F20>);
(<U1F29>,<U1F21>);(<U1F2A>,<U1F22>);(<U1F2B>,<U1F23>);(<U1F2C>,<U1F24>);
(<U1F2D>,<U1F25>);(<U1F2E>,<U1F26>);(<U1F2F>,<U1F27>);(<U1F38>,<U1F30>);
(<U1F39>,<U1F31>);(<U1F3A>,<U1F32>);(<U1F3B>,<U1F33>);(<U1F3C>,<U1F34>);
(<U1F3D>,<U1F35>);(<U1F3E>,<U1F36>);(<U1F3F>,<U1F37>);(<U1F48>,<U1F40>);
(<U1F49>,<U1F41>);(<U1F4A>,<U1F42>);(<U1F4B>,<U1F43>);(<U1F4C>,<U1F44>);
(<U1F4D>,<U1F45>);(<U1F59>,<U1F51>);(<U1F5B>,<U1F53>);(<U1F5D>,<U1F55>);
(<U1F5F>,<U1F57>);(<U1F68>,<U1F60>);(<U1F69>,<U1F61>);(<U1F6A>,<U1F62>);
(<U1F6B>,<U1F63>);(<U1F6C>,<U1F64>);(<U1F6D>,<U1F65>);(<U1F6E>,<U1F66>);
(<U1F6F>,<U1F67>);(<U1F88>,<U1F80>);(<U1F89>,<U1F81>);(<U1F8A>,<U1F82>);
(<U1F8B>,<U1F83>);(<U1F8C>,<U1F84>);(<U1F8D>,<U1F85>);(<U1F8E>,<U1F86>);
(<U1F8F>,<U1F87>);(<U1F98>,<U1F90>);(<U1F99>,<U1F91>);(<U1F9A>,<U1F92>);
(<U1F9B>,<U1F93>);(<U1F9C>,<U1F94>);(<U1F9D>,<U1F95>);(<U1F9E>,<U1F96>);
(<U1F9F>,<U1F97>);(<U1FA8>,<U1FA0>);(<U1FA9>,<U1FA1>);(<U1FAA>,<U1FA2>);
(<U1FAB>,<U1FA3>);(<U1FAC>,<U1FA4>);(<U1FAD>,<U1FA5>);(<U1FAE>,<U1FA6>);
(<U1FAF>,<U1FA7>);(<U1FB8>,<U1FB0>);(<U1FB9>,<U1FB1>);(<U1FBA>,<U1F70>);
(<U1FBB>,<U1F71>);(<U1FBC>,<U1FB3>);(<U1FC8>,<U1F72>);(<U1FC9>,<U1F73>);
(<U1FCA>,<U1F74>);(<U1FCB>,<U1F75>);(<U1FCC>,<U1FC3>);(<U1FD8>,<U1FD0>);
(<U1FD9>,<U1FD1>);(<U1FDA>,<U1F76>);(<U1FDB>,<U1F77>);(<U1FE8>,<U1FE0>);
(<U1FE9>,<U1FE1>);(<U1FEA>,<U1F7A>);(<U1FEB>,<U1F7B>);(<U1FEC>,<U1FE5>);
(<U1FF8>,<U1F78>);(<U1FF9>,<U1F79>);(<U1FFA>,<U1F7C>);(<U1FFB>,<U1F7D>);
(<U1FFC>,<U1FF3>);(<U2126>,<U03C9>);(<U212A>,<U006B>);(<U212B>,<U00E5>);
(<U2160>,<U2170>);(<U2161>,<U2171>);(<U2162>,<U2172>);(<U2163>,<U2173>);
(<U2164>,<U2174>);(<U2165>,<U2175>);(<U2166>,<U2176>);(<U2167>,<U2177>);
(<U2168>,<U2178>);(<U2169>,<U2179>);(<U216A>,<U217A>);(<U216B>,<U217B>);
(<U216C>,<U217C>);(<U216D>,<U217D>);(<U216E>,<U217E>);(<U216F>,<U217F>)
```

C.3 Additional case mappings

```
(<U0131>,<U0069>);(<U03C2>,<U03C3>)
```

Annex D
(informative)

Concepts

D.1 General

This annex describes major features in the language using examples of the use of the features and textual discussion on how the features function.

D.2 Files

A file is a collection of records that may be placed into or retrieved from a storage medium. The user not only chooses the file organization, but also chooses the file processing method and sequence. Although the file organization and processing method are restricted for sequential media, no such restrictions exist for mass storage media.

When describing the capabilities of COBOL to manipulate files, the following conventions are used. The term "file-name" means the user-defined word used to reference a file. The terms "file referenced by file-name" and "file" mean the physical file regardless of the file-name used in the COBOL program. The term "file connector" means the entity containing information concerning the file. All accesses to physical files occur through file connectors. In various implementations the file connector is referred to as a file information table, a file control block, etc.

D.2.1 File organization

D.2.1.1 Sequential organization

Sequential files are organized so that each record, except the last, has a unique successor record; each record, except the first, has a unique predecessor record. The successor relationships are established by the order of execution of WRITE statements when the file is created. Once established, successor relationships do not change except in the case where records are added to the end of a file.

A sequentially organized mass storage file has the same logical structure as a file on any sequential medium; however, a sequential mass storage file may be updated in place. When this technique is used, new records cannot be added to the file and each replaced record shall be the same size as the original record.

D.2.1.2 Relative organization

A file with relative organization is a mass storage file from which any record may be stored or retrieved by providing the value of its relative record number.

Conceptually, a file with relative organization is a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number. Each logical record in a relative file is identified by the relative record number of its storage area. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in any of the first through the ninth record areas.

In order to achieve more efficient access to records in a relative file, the number of bytes reserved on the medium to store a particular logical record may be different from the number of bytes in the description of that record in the data division.

D.2.1.3 Indexed organization

A file with indexed organization is a mass storage file from which any record may be accessed by giving the value of a specified key in that record. For each key data item defined for the records of a file, an index is maintained. Each such index represents the set of values from the corresponding key data item in each record. Each index, therefore, is a mechanism that may provide access to any record in the file.

Each indexed file has a primary index that represents the prime record key of each record in the file. Each record is inserted in the file, changed, or deleted from the file based solely upon the value of its prime record key. The prime record key of each record in the file shall be unique, and it shall not be changed when updating a record. The prime record key is declared in the RECORD KEY clause of the file control entry for the file.

Alternate record keys provide alternative means of retrieval for the records of a file. Such keys are named in the ALTERNATE RECORD KEY clauses of the file control entry. When the DUPLICATES phrase is specified in the ALTERNATE RECORD KEY clause, the value of a particular alternate record key need not be unique within the file.

D.2.1.4 Logical records

A logical record is the unit of data that is retrieved from or stored into a file. The number of records that may exist in a file is limited only by the capacity of the storage media. There are two types of records: fixed length and variable length. When a file is created, it is declared to contain either fixed-length or variable-length records. In any case, the content of the record area does not reflect any information the implementor may add to the record on the physical storage medium (such as record length headers), nor does the length of the record used by the COBOL programmer reflect these additions.

D.2.1.4.1 Fixed-length records

Fixed-length records shall contain the same number of bytes for all the records in the file. All input-output operations on the file may process only this one record size. Fixed-length records may be explicitly selected by specifying format 1 of the RECORD clause in the file description entry for the file regardless of the individual record descriptions.

D.2.1.4.2 Variable-length records

Variable-length records may contain differing numbers of bytes among the records on the file. To define variable-length records explicitly, the VARYING phrase may be specified in the RECORD clause in the file description entry or the sort-merge file description entry for the file. The length of a record is affected by the data item referenced in the DEPENDING phrase of the RECORD clause or the DEPENDING phrase of an OCCURS clause or by the length of the record description entry for the file.

D.2.1.4.3 Implementor-defined record types

Where no RECORD clause is specified in the file description entry for a file, or where the RECORD clause specifies a range of byte positions, it is implementor-defined whether fixed-length or variable-length records are obtained.

D.2.2 File processing

A file may be processed by performing operations upon individual records or upon the file as a unit. Unusual conditions that occur during processing are communicated back to the runtime element.

D.2.2.1 Record operations

The ACCESS MODE clause of the file description entry specifies the manner in which the runtime element operates upon records within a file. The access mode may be sequential, random, or dynamic.

For files that are organized as relative or indexed, any of the three access modes may be used to access the file regardless of the access mode used to create the file. A file with sequential organization may only be accessed in sequential mode.

The organization, format, and contents of an output report may be specified using the report writer feature. (See D.19, Report writer.)

D.2.2.1.1 Sequential access mode

A file may be accessed sequentially irrespective of the file organization.

For sequential organization, the order of sequential access is the order in which the records were originally written.

For relative organization, the order of sequential access is ascending based on the value of the relative record numbers. Only records that currently exist in the file are made available. The START statement may be used to establish a starting point for a series of subsequent sequential retrievals.

For indexed organization, the order of sequential access is ascending based on the value of the key of reference according to the collating sequence associated with the file. Any of the keys associated with the file may be established as the key of reference during the processing of the file. The order of retrieval from a set of records that have duplicate key of reference values is the original order of arrival of those records into the set. The START statement may be used to establish a starting point within an indexed file for a series of subsequent sequential retrievals.

Records can be accessed in the reverse order from which they are organized in the file by specifying the PREVIOUS phrase on the READ statement.

D.2.2.1.2 Random access mode

When a file is accessed in random mode, input-output statements are used to access the records in a programmer-specified order. The random access mode may only be used with relative or indexed file organizations.

For a file with relative organization, the programmer specifies the desired record by placing its relative record number in a relative key data item. With the indexed organization, the programmer specifies the desired record by placing the value of one of its record keys in a record key or an alternate record key data item.

D.2.2.1.3 Dynamic access mode

With dynamic access mode, the programmer may change at will from sequential accessing to random accessing, using appropriate forms of input-output statements. The dynamic access mode may only be used on files with relative or indexed organizations.

D.2.2.1.4 Open mode

The open mode of the file is related to the actions to be performed upon records in the file. The open modes and purposes are: input, to retrieve records; output, to place records into a file; extend, to append records to an existing file; I-O, to retrieve and update records. The open mode is specified in the OPEN statement.

When the open mode is input, a file may be accessed by the READ statement. The START statement may also be used for files organized as indexed or relative that are in sequential or dynamic access modes.

When the open mode is output, the records are placed into the file by issuing WRITE, GENERATE, or TERMINATE statements.

When the open mode is extend, new records are added to the logical end of a file by issuing WRITE, GENERATE, or TERMINATE statements.

Only mass storage files may be referenced in the open I-O mode. The additional capabilities of mass storage devices permit updating in place, thus READ and REWRITE statements may always be used. A mass storage file may be updated in the same manner as a file on a sequential medium, by transcribing the entire file into another file (perhaps in a separate area of mass storage) using READ and WRITE statements. However, it is sometimes

more efficient to update a mass storage file in place. This mass storage file maintenance technique uses the REWRITE statement to return to their previous locations on the storage medium only those records that have changed. READ and REWRITE statements are the only operations allowed while updating in place sequentially organized files. For files with relative or indexed organization, the following additional functions may be applied: the START statement may be used in sequential or dynamic access mode to alter the sequence of record retrieval; the DELETE statement may be used with any access mode to remove a record logically from a file; the WRITE statement may be used in random or dynamic access mode to insert a new record into the file.

D.2.2.1.5 Current volume pointer

The current volume pointer is a conceptual entity used in this document to facilitate exact specification of the current physical volume of a sequential file. The status of the current volume pointer is affected by the CLOSE, OPEN, READ, and WRITE statements.

D.2.2.1.6 File position indicator

The file position indicator is a conceptual entity used in this document to facilitate exact specification of the next record to be accessed within a given file during certain sequences of input-output operations. The setting of the file position indicator is affected only by a CLOSE statement with a REEL or UNIT phrase, an OPEN statement, a READ statement, and a START statement. The concept of a file position indicator has no meaning for a file opened in the output or extend mode.

D.2.2.1.7 Linage concepts

The LINAGE clause may be used when specifying an output report. It facilitates definition of a logical page, and the positioning within that logical page of top and bottom margins and a footing area. Use of the LINAGE clause implicitly defines an associated identifier, the LINAGE-COUNTER, that acts as a pointer to a line within the page body.

D.2.2.2 File operations

Several COBOL statements operate upon files as entities or as collections of records. These are the CLOSE, MERGE, OPEN, and SORT statements.

D.2.2.2.1 Sorting and merging

The SORT and MERGE statements organize records from one or more files into a sequence based upon specified keys that are data items within those records.

The SORT statement orders records from a file that need not have originally been in the required sequence. The MERGE statement combines records from two or more files, where the records of each file were already in the required sequence.

D.2.2.2.1.1 Sorting

In many sort applications it is necessary to apply some special processing to the contents of a sort file. The special processing may consist of addition, deletion, creation, altering, editing, or other modification of the individual records in the file. It may be necessary to apply the special processing before or after the records are reordered by the sort, or special processing may be required in both places. The COBOL sort feature allows the user to express these procedures and to specify at which point, before or after the sort, they are to be executed. A COBOL program may contain any number of sorts, and each of them may have its own input and output procedures. The sort feature automatically causes execution of these procedures at the specified point.

Within an input procedure, the RELEASE statement writes records to the sort file. That is, at the completion of execution of the input procedure those records that have been processed by use of the RELEASE statement (rather than the WRITE statement) constitute the sort file, and this file is available only to the SORT statement. Execution of the SORT statement arranges the entire set of records in the sort file according to the keys specified in the SORT statement. The sorted records are made available from the sort file by use of the RETURN statement during

execution of the output procedure or by automatically writing the records to one or more files by the use of the GIVING phrase.

D.2.2.2.1.2 Merging

In some applications it is necessary to apply some special processing to the contents of a merged file. The special processing may consist of addition, deletion, altering, editing, or other modification of the individual records in the file. The COBOL merge feature allows the user to express an output procedure to be executed as the merged output is created. The merged records are made available from the merge file by use of the RETURN statement in the output procedure or by automatically writing the records to one or more files by the use of the GIVING phrase.

D.2.2.3 Exception handling

During the execution of any input or output operation, unusual conditions may arise that preclude normal completion of the operation. There are four methods by which these conditions are communicated to a runtime element; a data item associated with the FILE STATUS clause, exception declarative, exception functions, and optional phrases associated with the imperative statement.

D.2.2.3.1 I-O status

I-O status is a conceptual entity used in this document to facilitate exact specification of the status of the execution of an input-output operation. The setting of I-O status is affected only by the CLOSE, DELETE, OPEN, READ, REWRITE, START, WRITE, and UNLOCK statements. The I-O status value for a given file is made available to a runtime element via the data item referenced by the data-name specified in the FILE STATUS clause of the file description entry for that file and via the EXCEPTION-FILE or EXCEPTION-FILE-N function. The I-O status value is placed into this data item or made available to the EXCEPTION-FILE and EXCEPTION-FILE-N functions during the execution of the input-output statement and prior to the execution of any imperative statement associated with that input-output statement or prior to the execution of any exception declarative.

D.2.2.3.2 Exception declaratives

A USE AFTER EXCEPTION procedure, when one is specified for the file, is executed whenever an input or output condition arises that results in an unsuccessful input-output operation. However, the exception declarative is not executed if the condition is invalid key and the INVALID KEY phrase is specified, or if the condition is at end and the AT END phrase is specified.

D.2.2.3.3 Exception functions

The exception function EXCEPTION-FILE or EXCEPTION-FILE-N may be referenced in a statement in the logical range of an exception declarative in order to determine the I-O status value resulting from the input-output operation that caused the declarative to be executed as well as the associated file-name. In addition, the exception function EXCEPTION-STATUS may be referenced to determine the exception-name that was returned (such as EC-I-O-AT-END).

D.2.2.3.4 Optional phrases

The INVALID KEY phrases may be associated with the DELETE, READ, REWRITE, START, or WRITE statements. Some of the conditions that give rise to an invalid key condition are when a requested key does not exist in the file (DELETE, READ, or START statements), when a key is already in a file and duplicates are not allowed (WRITE statement), and when a key does not exist in the file or when it was not the last key read (REWRITE statement). If the invalid key condition occurs during the execution of a statement for which the INVALID KEY phrase has been specified, the statement identified by that INVALID KEY phrase is executed.

The AT END phrase may be associated with a READ statement. The at end condition occurs in a sequentially accessed file when no next logical record exists in the file, when the number of significant digits in the relative record number is larger than the size of the relative key data item, or when an optional file is not present. If the at end condition occurs during the execution of a statement for which the AT END phrase has been specified, the statement identified by that AT END phrase is executed.

D.2.3 File sharing and record locking

File sharing and record locking provide a way to ensure integrity of data in a physical file being accessed by multiple logical files simultaneously. Integrity is ensured only when all accesses to the physical file appropriately utilize file sharing and record locking conventions.

File sharing provides the capability of sharing a physical file among different logical files by specifying the SHARING clause in the file control entry or the SHARING phrase in an OPEN statement.

Record locking provides the capability of managing concurrent access to records in a shared physical file. Record locking is selected by specifying the LOCK MODE clause in the file control entry.

The rules for file sharing and record locking are the same whether the physical file is shared among different run units or different runtime elements in a COBOL run unit or different file definitions within a COBOL runtime element.

Implementation of file sharing and record locking such that concurrent access is managed for both COBOL and non-COBOL runtime elements is recommended; however, the requirement and specification of this is beyond the scope of the COBOL specification. This final committee draft International Standard describes file sharing and record locking in a COBOL context.

Transaction processing facilities, such as commit and rollback, are not included in this final committee draft International Standard.

D.2.3.1 File sharing

A physical file can be shared only when it resides on a medium that allows concurrent access to the physical file.

File sharing is selected for access to a given physical file by coding a SHARING clause in the file control entry that describes the logical file or in an OPEN statement that opens the logical file. A sharing mode specified in an OPEN statement overrides a sharing mode specified in a file control entry. Permitted sharing and input-output operations are further specified by the i-o modes of OPEN statements.

Each OPEN statement succeeds or fails on the basis of the most restrictive sharing mode and i-o mode already in effect for the physical file as well as the sharing mode and i-o mode specified for that OPEN statement. The permitted sharing, then, is dynamic based on the sharing and i-o modes of the on-going opening and closing of logical files. When a shared file is already open, a subsequent OPEN statement that specifies the OUTPUT open mode for that file will be unsuccessful.

The specification for the OPEN statement in Table 20, Opening available shared files that are currently open by another file connector, should be reviewed for an understanding of the detailed interaction of opening request and sharing modes. The following is a simplified description:

— SHARING WITH NO OTHER establishes exclusive access; the physical file cannot be opened by any other logical file. The opening of a logical file in this sharing mode will fail if any other logical file already has the physical file open.

— SHARING WITH READ ONLY establishes that the physical file may be opened by another logical file for input, provided that the other logical file's sharing mode permits the sharing and i-o mode of this OPEN statement. The opening of a file with SHARING WITH READ ONLY will fail if another logical file has the file open in a more restrictive sharing mode or i-o mode. Record locking capabilities should be used to control concurrent access so that retrieved records reflect correct content.

— SHARING WITH ALL OTHER establishes that the physical file may be opened by other logical files in sharing mode READ ONLY or in sharing mode ALL OTHER and in any i-o mode. Concurrent access is possible both for update and retrieval, and for retrieval of records that are in process of being updated. Record locking capabilities should be used to control concurrent access so that updates are not overwritten or lost and that retrieved records reflect correct content.

To allow compatible behavior with the previous COBOL standard, default sharing, if any, is defined by the implementor.

D.2.3.2 Record locking

Record locking gives application developers the ability to control concurrent access to logical records in a shared physical file. Concurrent access is possible when a physical file is opened by more than one logical file.

In order to guarantee the integrity of records in a physical file, it is usually necessary for applications to restrict access to records being updated or deleted — but this depends on the nature of the application. The mechanism for restricting access to a record is called a record lock.

COBOL provides two modes of record locking — automatic locking and manual locking — that can be selected by coding a LOCK MODE clause in a file control entry. The following combinations of access mode and lock mode are provided:

Access mode	Lock mode	Single/multiple records
Sequential	Automatic or manual	Single only
Random or dynamic	Automatic or manual	Single or multiple

When sharing WITH NO OTHER is specified, record locking is not applicable and use of any syntax that requests record locking is ignored.

D.2.3.2.1 Automatic locking

LOCK MODE AUTOMATIC indicates that the runtime system will take care of locking records. When automatic locking is in effect, the time at which locks are acquired and released is defined by the COBOL specification, rather than controlled by the application. At the same time, however, other logical files can access the physical file with either manual locking or automatic locking.

It is easier for an application to control concurrent access to records with automatic locking than with manual locking, but it might not meet the needs of all applications.

Single record or multiple record automatic locking can be selected.

With single record automatic locking, a logical file has a lock on one record at a time. The successful execution of a READ statement establishes a lock on the newly read record. The execution of a READ, REWRITE, WRITE, DELETE, UNLOCK, or CLOSE statement releases a lock on a previously-locked record.

With multiple record automatic locking, all newly read records are automatically locked and the lock for a record is retained until the record is deleted, or an UNLOCK statement is executed for the logical file, or the logical file is closed.

The following illustrates a file-control paragraph that defines a logical file (my-file), where the physical file (accounts) resides on a mass storage device. SHARING WITH READ ONLY indicates that the physical file can be shared with other logical files as long as those files are open for READ ONLY. The logical file my-file is not itself restricted to read only.

```
FILE-CONTROL.  SELECT my-file   ASSIGN TO accounts
...
SHARING WITH READ ONLY
LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORDS.
```

Single record automatic locking is selected by the following lock mode clause:

```
LOCK MODE IS AUTOMATIC
```

The automatic acquisition and release of record locks is summarized in Table C.1 —, Summary of record lock acquisition and release.

D.2.3.2.2 Manual locking

LOCK MODE MANUAL indicates that whether and when locks are acquired and released is completely under control of the application; thus, data integrity is completely under application control. Manual locking provides the flexibility to select locking only for records that require it and to release locks at convenient times. The use of manual locking requires careful design of all the applications that share access to a given physical file.

An application can manually lock a single record at a time or multiple records at a time.

With single record manual locking, each record lock is established by specifying the LOCK phrase on an I/O statement and the lock is automatically released on the next execution of a READ, REWRITE, WRITE, DELETE, UNLOCK, or CLOSE statement — in the same manner as for single record automatic locking.

With multiple record manual locking, each record lock is established by specifying the LOCK phrase on an I/O statement and all locks are held until explicitly released. Successful execution of an UNLOCK statement or a CLOSE statement releases all locks. Successful execution of an I/O statement with a NO LOCK phrase releases a lock on the record that is processed by that I/O statement. Successful execution of a DELETE statement releases a lock on the deleted record.

Additional options on I/O statements allow for selective actions that are not normally needed, but may be useful in special circumstances:

```
READ ... ADVANCING ON LOCK:
```

Locked records are skipped and the next unlocked record in sequential order is retrieved.

```
READ ... IGNORING LOCK:
```

A record is retrieved even if it is locked.

The following illustrates a file-control paragraph that defines a logical file (my-file), where the physical file (accounts) resides on a mass storage device. SHARING WITH ALL OTHER indicates that the physical file can be shared with other logical files and any of them can update records.

```
FILE-CONTROL.  SELECT my-file   ASSIGN TO accounts
...
SHARING WITH ALL OTHER
LOCK MODE IS MANUAL WITH LOCK ON MULTIPLE RECORDS.
```

The manual locking and unlocking of records is controlled by options on I/O statements, by the UNLOCK statement, and by the CLOSE statement, as summarized in table C.1, Summary of record lock acquisition and release.

Table C.1 — Summary of record lock acquisition and release

I-O statement	Lock mode	Lock is set	Lock for same logical file is released
DELETE	Automatic - single	no	- on successfully deleted record, if locked; - otherwise on any previously locked record
	-multiple	no	- on the deleted record, if locked
	Manual - single	no	-on successfully deleted record, if locked; -otherwise on any previously locked record
	-multiple	no	- on the deleted record, if locked
CLOSE	any	no	- on all locked records
READ	Automatic - single	- on the retrieved record	- on any previously locked record
	- multiple	- on the retrieved record	no
	Manual -single	- on the retrieved record if LOCK specified	- on any previously locked record
	- multiple	- on the retrieved record if LOCK specified	- on the retrieved record, if locked and NO LOCK is specified
OPEN	any	no	no
REWRITE	Automatic - single	no	- on successfully rewritten record if locked - otherwise, on any previously locked record
	- multiple	no	no
	Manual - single	- on the rewritten record if LOCK specified	- on successfully rewritten record if locked, unless LOCK is specified - and on any previously locked record
	- multiple	- on the rewritten record if LOCK specified	- on the rewritten record if locked and NO LOCK is specified
START	any	no	no
WRITE	Automatic - single	no	- on any previously locked record
	- multiple	no	no
	Manual - single	- if LOCK is specified	- on any previously locked record
	- multiple	- if LOCK is specified	no

UNLOCK	Automatic -single - multiple Manual - single - multiple		for all lock modes: - on all locked records

The LOCK phrase of a CLOSE statement is unrelated to record locking.

D.2.3.3 Retry

The RETRY phrase gives the capability of indicating a willingness to wait to obtain a locked file or record. The RETRY phrase may specify a waiting period in seconds or the number of times to retry the operation. The number of times to retry can be a number or the word FOREVER, which implies that the operating environment attempts to retry the operation until some external occurrence terminates the task or the locked record or file becomes available. If the lock condition exists throughout, or if there is a lock but no RETRY phrase is specified, the operating environment reports the file sharing conflict condition or record operation conflict condition by setting the I-O status value associated with the file connector to the appropriate value.

D.3 Tables and any-length elementary items

Tables of data are common components of business data processing problems. Although the repeating items that make up a table could be otherwise described by a series of separate data description entries all having the same level-number and all subordinate to the same group item, there are two reasons why this approach is not satisfactory. First, from a documentation standpoint, the underlying homogeneity of the items would not be readily apparent; and second, the problem of making available an individual element of such a table would be severe when there is a decision as to which runtime element is made available.

A table of data items is defined by coding the OCCURS clause in its data description entry. This clause specifies that the item is to be repeated as many times as stated in the OCCURS clause. The number of occurrences of a table element may be specified to be fixed or variable. The item is considered to be a table element and its name and description apply to each occurrence. The collection of all these elements is the table itself.

A reference to a specific occurrence of the table elements can be made only by specifying the data-name of the table element, including any necessary qualifiers, followed by the desired occurrence number. The occurrence number is known as a subscript.

D.3.1 Table definition

To define a table, code the OCCURS clause in the data description of the table element. The table so defined is the collection of all elements described with that OCCURS clause, including all items subordinate to these elements.

Example 1 shows the data description entry of the table element INDICATOR, and the reference to the twelfth element of the table so defined in a MOVE statement. Each element of that table consists of the elementary item INDICATOR. There are twenty table-elements INDICATOR. However, it is not possible to refer to the table as a unit.

Example 1

```
01  FULL-RECORD.
    02  ANOTHER-ITEM  PIC X(10).
    02  INDICATOR  PICTURE XXX OCCURS 20 TIMES.

MOVE SPACES TO INDICATOR (12)
```

To reference the complete table, define the table as the only element of a group item. Example 2 shows the definition of the same table as shown in example 1, which is now part of the group item INDICATOR-TABLE, and the use of that group item in a MOVE statement.

Example 2

```
01  FULL-RECORD.
    02  ANOTHER-ITEM  PIC X(10).
    02  INDICATOR-TABLE.
        03  INDICATOR  PICTURE XXX OCCURS 20 TIMES.

    MOVE SPACES TO INDICATOR-TABLE
```

Example 3 shows a table defined by the item MONTHLY-REVENUES. Each table-element is a group item consisting of the two elementary items REV-SHOP-1 and REV-SHOP-2. A reference to the group item REVENUES-TABLE refers to the complete table MONTHLY-REVENUES.

Example 3

```
01  REVENUES-TABLE.
    02  MONTHLY-REVENUES   OCCURS 12 TIMES.
        03 REV-SHOP-1 PIC S9(8)V99.
        03 REV-SHOP-2 PIC S9(8)V99.
```

Example 4 shows two tables that are part of the same group item. A reference to the group item TABLES refers to the complete table MONTHLY-REVENUES combined with the complete table MONTHLY-PROFITS.

Example 4

```
01  TABLES.
    02  MONTHLY-REVENUES   OCCURS 12 TIMES.
        03 REV-SHOP-1 PIC S9(8)V99.
        03 REV-SHOP-2 PIC S9(8)V99.
    02  MONTHLY-PROFITS    OCCURS 12 TIMES.
        03 PRO-SHOP-1 PIC S9(8)V99.
        03 PRO-SHOP-2 PIC S9(8)V99.
```

If any group item containing a table is also described with the OCCURS clause, the table is said to be multi-dimensional; otherwise, the table is said to be one-dimensional. All tables in the previous examples are one-dimensional tables.

Example 5 shows a two-dimensional table. Each of the 12 occurrences of MONTHLY-REVENUES is a group item containing 100 occurrences of the elementary item REV-SHOP.

Example 5

```
01  A-TABLE.
    02  MONTHLY-REVENUES    OCCURS 12 TIMES.
        03 REV-SHOP PIC S9(8)V99   OCCURS 100 TIMES.
```

In the general case, to define an n-dimensional table, code the OCCURS clause in the data description of the element of the table and in the descriptions of (n – 1) group items that contain the element.

D.3.2 Values of tables

In the working-storage and local-storage sections, initial values of elements within tables are specified in one of the following ways:

1) The table may be described as a series of separate data description entries all subordinate to the same group item, each of which specifies the value of an element, or part of an element, of the table. In defining the record and its elements, any data description clause (USAGE, PICTURE, etc.) may be used to complete the definition, where required. The hierarchical structure of the table is then shown by use of the REDEFINES entry and its associated subordinate entries. The subordinate entries, following the REDEFINES entry, that are repeated due to OCCURS clauses, shall not contain VALUE clauses.

2) All the dimensions of a table may be initialized by associating the VALUE clause with the description of the entry defining the entire table. The lower level entries will show the hierarchical structure of the table; lower level entries shall not contain VALUE clauses.

3) The value of selected table elements may be specified using VALUE clauses with or without the optional TO phrase.

D.3.3 References to table items

Whenever the user references a table element or a condition-name associated with a table element, the reference shall indicate which occurrence of the element is intended, except in a SEARCH statement, in a SORT statement, and in arguments to certain intrinsic functions that reference a table. For access to a one-dimensional table the occurrence number of the desired element provides complete information. For tables of more than one dimension, an occurrence number shall be supplied for each dimension of the table. In example 5, then, a reference to MONTHLY-REVENUES (3) is complete, but a reference to REV-SHOP (2) is not since any reference to a REV-SHOP item requires two subscripts. A valid reference might be REV-SHOP (5, 2) which refers to the second occurrence of REV-SHOP in the fifth occurrence of MONTHLY-REVENUES.

D.3.4 Subscripting

Occurrence numbers are specified by appending one or more subscripts to the data-name.

The subscript may be represented either by an arithmetic-expression that produces an integer result or by an index-name associated with the table. An index-name may be followed by either the operator + or the operator – and an integer, which is used as in increment or decrement, respectively. It is permissible to mix both arithmetic-expressions and index-names in a single subscript list.

The subscripts, enclosed in parentheses, are written immediately following any qualification for the name of the table element. The number of subscripts in such a reference shall equal the number of dimensions in the table whose element is being referenced. That is, there shall be a subscript for each OCCURS clause in the hierarchy containing the data-name including the data-name itself.

When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization. If a multi-dimensional table is thought of as a series of nested tables and the most inclusive or outermost table in the nest is considered to be the major table with the innermost or least inclusive table being the minor table, the subscripts are written from left to right in the order major, intermediate, and minor.

A reference to an item shall not be subscripted if the item is not a table element or an item or condition-name within a table element.

The lowest permissible occurrence number is 1. The highest permissible occurrence number is the maximum number of occurrences of the item as specified in the OCCURS clause. A subscript value that exceeds the maximum number of occurrences or is negative or zero may be detected by using exception declaratives.

D.3.4.1 Subscripting using index-names

In order to facilitate such operations as table searching and manipulating specific items, a technique called indexing is available. To use this technique, the programmer assigns one or more index-names to an item whose data description entry contains an OCCURS clause. An index associated with an index-name acts as a subscript, and its value corresponds to an occurrence number for the item to which the index-name is associated.

The INDEXED BY phrase, by which the index-name is identified and associated with its table, is an optional part of the OCCURS clause. There is no separate entry to describe the index associated with index-name since its definition is completely hardware oriented. At runtime the contents of the index correspond to an occurrence number for that specific dimension of the table with which the index is associated; however, the manner of correspondence is determined by the implementor. The initial value of an index at runtime is undefined, and the index shall be initialized before use. The initial value of an index is assigned with the PERFORM statement with the VARYING phrase, the SEARCH statement with the ALL phrase, or the SET statement.

The use of an arithmetic-expression or data-name as a subscript referencing a table element or an item within a table element does not cause the alteration of any index associated with that table.

An index-name may be used to reference only the table to which it is associated via the INDEXED BY phrase.

Data that is arranged in the form of a table is often searched. The SEARCH statement provides facilities for producing serial and nonserial (for example, binary) searches. It is used to search a table for a table element that satisfies a specific condition and to adjust the value of the associated index to indicate that table element.

Relative indexing is an additional option for making references to a table element or to an item within a table element. When the name of a table element is followed by a subscript of the form (index-name + or - integer), the occurrence number required to complete the reference is the same as if index-name were set up or down by integer via the SET statement before the reference. The use of relative indexing does not cause the value of the index to be altered.

The value of an index may be made accessible to a runtime element by storing the value in an index data item. Index data items are described by a data description entry containing a USAGE INDEX clause. The index value is moved to the index data item by the execution of a SET statement.

D.3.4.2 Subscripting example

Assuming the following data definition:

```
02 XCOUNTER PIC S99.
02 YCOUNTER PIC S99.

02 BAKER  OCCURS 20 TIMES  INDEXED BY BAKER-INDEX ...
   03 CHARLIE ...
   03 DOG  OCCURS 5 TIMES ...
      04 EASY
         88 MAX  VALUE IS ...
      04 FOX ...
         05 GEORGE OCCURS 10 TIMES ...
            06 HARRY ...
            06 JIM ...
```

references to BAKER and CHARLIE require only one subscript, references to DOG, EASY, MAX, and FOX require two, and references to GEORGE, HARRY, and JIM require three.

To illustrate the requirement of order from major to minor, HARRY (18, 2, 7) means the HARRY in the seventh GEORGE, in the second DOG, in the eighteenth BAKER.

Mixing arithmetic-expressions and index-names is illustrated by the following:

```
HARRY (BAKER-INDEX – 3, 4, (XCOUNTER * 2) – 3).
```

The use of an arithmetic expression that starts with a unary operator and follows an identifier is illustrated by the following:

```
DOG (XCOUNTER (- YCOUNTER))
```

Omitting the inner pair of parentheses, the identifier will be interpreted as:

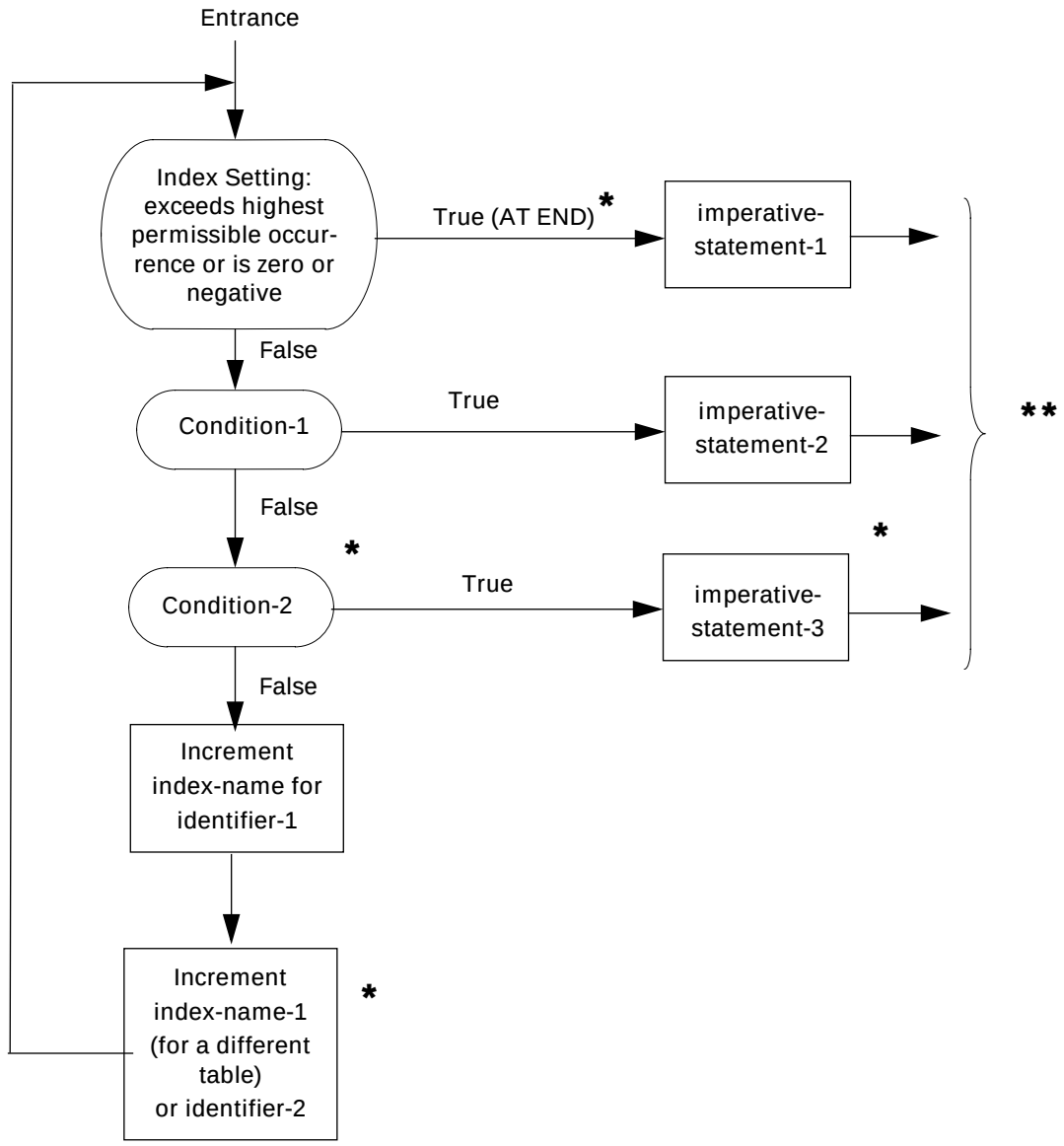
```
DOG (XCOUNTER - YCOUNTER)
```

which will cause a syntax error, because only one subscript is coded where two subscripts are necessary.

D.3.4.3 SEARCH example

A representation of the action of a serial format of a SEARCH statement containing two WHEN phrases is shown below. This figure is not intended to dictate implementation. The sample statement would be:

```
SEARCH table AT END imperative-statement-1
  WHEN condition-1 imperative-statement-2
WHEN condition-2 imperative-statement-3
END-SEARCH
```



*These operations are options included only when specified in the SEARCH statement.

**Each of these control transfers is to the end of the SEARCH statement unless the imperative-statement ends with a GO TO statement.

Figure C.1 — Format 1 SEARCH statement having two WHEN phrases

D.3.5 Sorting tables

The SORT statement can be used to sort tables into a user-defined order. This is especially useful for tables used with SEARCH ALL. The following examples illustrate this capability.

D.3.5.1 Example 1

This example is a simple sort in which the table is sorted in order using the key definitions in the OCCURS clause of data item tabl to specify the sequence, that is elem-item2 is the major key (ascending) and elem-item1 is the secondary key (descending). It is then possible to use a SEARCH ALL statement knowing that all the elements are in the required order.

```
01 group-item.
       05  tabl occurs 10 times
              indexed by ind
              ascending elem-item2
              descending elem-item1.
           10  elem-item1 pic x.
           10  elem-item2 pic x.
    ...
       move "l3n3m3p3o3x1x1x1x1x1" to group-item.
       sort tabl.
       search all tabl
          at end
             display "not found"
          when elem-item1 (ind) = "m"
             if (elem-item1 (ind – 1) = "n")
             and (elem-item1 (ind + 1) = "l")
                display "elem-item1 is descending order – 2nd key"
             else
                display "sort failed"
             end-if
       end-search.
```

D.3.5.2 Example 2

This example is a simple sort in which the table is sorted in ascending order using each entire element of the table (data item tabl) to determine the sequence.

```
working-storage section.
01  group-item.
    05  tabl occurs 10 times.
        10  elem-item1 pic x.
        10  elem-item2 pic x.
...
procedure division.
...
    sort tabl ascending.
    if  tabl (1) ...
```

D.3.5.3 Example 3

This example is a sort in which the table is sorted based on specified key data items. The major key is elem-item2 even though it is not specified as a KEY in the OCCURS clause. The secondary is elem-item3. It is treated as a descending key for this sort because the DESCENDING (which is transitive across KEY data items) specified in the SORT statement takes precedence over the ASCENDING specified in the OCCURS clause.

```
working-storage section.
01  group-item.
    05  tabl occurs 10 times
           ascending elem-item3
           descending elem-item1.
        10  elem-item1 pic x.
        10  elem-item2 pic x.
```

```
           10 elem-item3 pic x.
      ...
      procedure division.
      ...
          sort tabl descending elem-item2 elem-item3.
          if tabl (1)...
```

D.3.5.4 Example 4

This example sorts only the third instance of tabl2, that is tabl2 (3). It uses the qualified data item, elem-item1 of group2 as its key. In normal procedure division references, elem-item1 of group2 requires two levels of subscripting/indexing while in this reference it has none. Similarly, tabl2 normally requires two levels of subscripting, but can not be subscripted as data-name-2 in the SORT statement. Instead it uses the value of t1-ind for determining which instance is sorted.

```
      working-storage section.
      01  group-item.
          05  tabl1 occurs 10 times
                     indexed by t1-ind t2-ind.
              10  tabl2 occurs 5 times.
                  15  group1.
                      20  elem-item1 pic x.
                  15 group2.
                      20  elem-item1 pic 9.
      ...
      procedure division.
      ...
          set t1-ind to 3.
          sort tabl2 (t1-ind) descending elem-item1 of group2.
          if group1 (3 1) ...
```

Note that the following is also acceptable syntax for this sort statement.

```
   sort tabl2 (t1-ind, ALL) descending elem-item1 of group2.
```

D.3.6 Dynamic-capacity tables

A dynamic-capacity table (or "dynamic table") is a table whose physical size, known as its "capacity", grows dynamically as you add more entries to it. Its maximum capacity is limited only by the resources your implementation can make available. Its capacity can also be reduced. To improve the description and the planning of memory resources, you can specify a minimum capacity and an expected capacity. If the expected capacity is reached, you receive a warning in the form of a nonfatal exception but, unless you terminate the process, further entries will continue to be added.

You define a dynamic table by coding an OCCURS clause with the keyword DYNAMIC. In the following example, any number of families can be stored, each having up 10 children. (If some family has more than 10 children, this is still allowed, but you can choose to signal a warning.)

```
   01  family-record.
     03  family-code      PIC X(10).
     03  family  OCCURS DYNAMIC
                 CAPACITY IN family-count
                 INITIALIZED.
       05  family-name    PIC X(30).
       05  childs-name    PIC X(30)
           OCCURS DYNAMIC TO 10
           CAPACITY IN child-count.
       05  family-zipcode PIC X(20).
     03  family-town      PIC X(20).
```

If Emily is born to family number 300 which previously had 2 children, you only need to code 3 as a subscript and a new entry is automatically created:

```
MOVE "Emily" TO childs-name (300, 3)
```

You can SORT and SEARCH a dynamic table (if it has a key) just like a fixed-capacity table.

To know the current capacity (number of entries) of the table at any time, include the CAPACITY phrase, as in the example. The capacity may be changed using the SET statement, as in:

```
      SET family-count TO 300
or    SET family-count UP BY 10
```

Any new entries created, and any unreferenced data in them, can automatically be initialized by including the INITIALIZED keyword in the OCCURS clause, as in the example.

An entire dynamic table may be moved to another table as part of a group MOVE. However, the tables defined in the two groups must coincide and have the same entry length. You can create a dynamic table from a fixed or OCCURS DEPENDING table and vice versa, because this type of group MOVE allows one of the two tables to be non-dynamic.

The storage needed for a dynamic table is allocated and released entirely automatically. If there are other data items adjacent to a dynamic table, their positions are not affected. For example, "family-code" and "family-town" in the example stay in the same fixed locations.

The initial capacity of a dynamic-capacity table may be defined using the VALUE clause with a TO phrase and several operands:

```
01  town-record.
  03  town-name   PIC X(20) OCCURS DYNAMIC FROM 1 TO 20
      VALUES ARE "Leeds", "Bordeaux", "Pisa" FROM (1) TO (3).
```

sets the initial capacity of the table to 3.

Dynamic-capacity tables may be used in all contexts and applications where a fixed-capacity table could be used, except in the file section.

D.3.7 Any-length elementary items

A truly any-length elementary item of unlimited size can be defined by coding the following picture format:

```
05  notes       PIC X ANY LENGTH.
```

If the item has a maximum length (n), ANY LENGTH LIMIT n is written or, if there is no theoretical maximum length, ANY LENGTH is sufficient.

Any-length items may be national items, or bit items:

```
05  city-name       PIC 1 ANY LENGTH.
05  customer-name   PIC N ANY LENGTH.
```

When a new value is stored in an any-length elementary item, the item's length is automatically adjusted, subject to any maximum length. For example,

```
MOVE "This product is no longer available" TO notes
```

sets the length of notes to 35. To find this length, you use

```
FUNCTION LENGTH (notes)
```

Until its value changes again, this data item will now behave exactly like a fixed-length item with a PICTURE of X(35), and all the statements normally available to an alphanumeric or national data item, such as MOVE, STRING, UNSTRING, INSPECT, ACCEPT, DISPLAY can be used with it.

Either of the following two statements

```
INITIALIZE notes    or     MOVE SPACES TO notes
```

reduces the item to a zero length data item.

The value is always stored contiguously but, because the length may vary, the physical location of the item may change from time to time. For this reason, you cannot obtain the ADDRESS of the item, although this may be used indirectly, as in a CALL or INVOKE. The value may be stored contiguously with its neighbors. When this is the case, the item is called a direct any-length elementary item (or a direct elementary item). Items that are at the 01 level, or specify the LIMIT phrase default to direct any-length elementary items. When the value is stored elsewhere, it is referred to as an indirect any-length elementary item. The INDIRECT phrase may be used to override the default for items that would be direct elementary items.

Any-length elementary items may be defined within any data structure, may be combined with other fixed-length or variable-length items, and may be moved as part of a group item, provided that the variable-length items are located in the same relative positions in both groups.

An any-length elementary item can be passed as an argument to an intrinsic function, unless the function specifies a fixed-length argument.

The structure of an any-length elementary item can be defined using the PREFIXED and DELIMITED phrases. These phrases are not mutually exclusive. The PREFIXED phrase creates a length field before the any-length elementary item. This field may be BINARY-CHAR, BINARY-SHORT, or BINARY-LONG. If not specified, BINARY-SHORT is assumed. The DELIMITED phrase inserts the specified delimiter after the any-length elementary item. If the any-length elementary item is specified in the FILE section, it must include one or both of these phrases in its definition.

D.4 Shared memory area

This feature is basically oriented toward saving runtime memory space as it allows more than one file to share the same file area and input-output areas.

If the record-area format of the SAME clause is specified, only the record area is shared and the input-output areas for each file remain independent. In this case any number of the files sharing the same record area may be active at one time. This factor may give rise to an increase in the speed of the run unit.

To illustrate this point, consider file maintenance. If the programmer assigns the same record area to both the old and new files, he not only saves memory, but because this technique eliminates a move of each record from the input to the output area, significant time savings result. An additional benefit of this technique is that the programmer need not define the record in detail as a part of both the old and new files. Rather, he defines the record completely in one case and simply includes the level 01 entry in the other. Because these record areas are in fact the same area, one set of names suffices for all processing requirements without requiring qualification.

If a format other than the record-area format of the SAME clause is specified, not only the record area but the input-output areas as well are shared.

As a result, only one of the files sharing the same set of areas is permitted to be active at one time. This form of the clause is designed for the application in which a series of files is used during different phases of the run unit. In these cases, the SAME clause allows the programmer to save memory space.

D.5 Sharing of storage among data items

Through the REDEFINES clause and through the implicit redefinition of records associated with file descriptions, it is possible for the same storage area to be defined in different ways. When the same storage area is used for different purposes during the execution of a given statement, there may be unexpected results.

This applies when the data items associated with the storage area are explicitly referenced in the general formats and syntax rules of the statement. 14.6.9, Overlapping operands, addresses this point directly. It also applies when

the item is identified in the rules for the procedure division statement as being referenced or modified by it, even when the general format for the statement does not provide for the inclusion of the name of the item in the statement text.

The user is cautioned that the contents of the data items in such cases may be unexpected.

As an example, consider the following program fragment:

```
SELECT AFile ASSIGN USING AssignField
ORGANIZATION IS RELATIVE
ACCESS MODE IS SEQUENTIAL
FILE STATUS IS FileStatField
RELATIVE KEY IS RelKeyField.

FD  AFILE RECORD CONTAINS 80 CHARACTERS.

WORKING-STORAGE SECTION.
01 WS-Rec PICTURE X(80).
01 WS-Redef-1 REDEFINES WS-Rec.
   05 AssignField PICTURE X(10).
01 WS-Redef-2 REDEFINES WS-Rec.
   05 FileStatField PICTURE X(2).
01 WS-Redef-3 REDEFINES WS-Rec.
   05 RelKeyField PICTURE 9(4) USAGE PACKED-DECIMAL.

READ AFile INTO WS-Rec.
```

The data items FileStatField and RelKeyField are “implicitly referenced” by virtue of the discussion of the impact of the READ statement on their contents in the general rules for READ.

AssignField is not mentioned in the general formats, the syntax rules or the general rules for the READ statement at all.

The content of all three of these data items will almost certainly not be what is "expected" given their intended purposes after execution of a successful READ statement, because it is probable that the following events are likely to have occurred in something approximating the following order:

1) FileStatField is set to a value of "00" during the I/O operation.

2) At the conclusion of the I/O operation, RelKeyField is set to the relative record number of the record just read.

3) After the I/O operation, the contents of the record area associated with AFile are MOVEd to WS Rec, overwriting AssignField (and FileStatField and RelKeyField).

Although these results are well-defined, it is unlikely that the content of FileStatField or RelKeyField will reflect meaningful values at the completion of the READ, and it is also probable that the content of AssignField will have been changed from its original implementor-defined value as part of the implicit MOVE.

A second example where this principle applies to multiple receiving operands (without reference to any sending operands) can be found in the multiple-destination COMPUTE statement:

```
01  Receiving-rec.
    03  Map-1.
        05  Res-1 PIC V9(8).
        05  FILLER PIC XX.
    03  Map-2 REDEFINES Map-1.
        05  FILLER PIC X.
        05  Res-2 PIC V9(8).
    03  Map-3 REDEFINES Map-2.
        05  FILLER PIC XX.
        05  Res-3 PIC V9(8).

COMPUTE Res-1 Res-2 Res-3 = FUNCTION RANDOM.
```

In this instance, the result of execution is well-defined; however, it is unlikely that the content of either Res-1 or Res-2 will be particularly useful, nor will they represent the same values as Res-3 as would be the case if the operands shared no part of their storage areas.

A third example of circumstances in which there are multiple receiving operands specified, with results that might not be expected, can be found in the CALL statement:

```
01  Parameter-rec.
    03  Params.
        05  FILLER PIC XXX.
        05  Param-1 PIC X(5).
    03  Params-redef-1 PIC XX.
        05  FILLER PIC XX.
        05  Param-2 PIC X(5)
    03  Params-redef-2 PIC XX.
        05  FILLER PIC X.
        05  Param-2 PIC X(5).

CALL "Some-Prog" USING REFERENCE Param-1 Param-2 Param-3.
```

In this case, the three by-reference parameters are treated as receiving operands by the rules of the CALL statement, but the program executing the CALL statement has no control over whether or not "Some-Prog" updates any, much less all, of these parameters, or over the order in which they are updated. What parameters contain information that is of significance is unclear in such a circumstance.

The fourth example is a slight modification of the second:

```
CALL "Some-Prog" USING REFERENCE Param-1 Param-2 Param-3
RETURNING Parameter rec.
```

In this case, whether or not "Some-Prog" updates any or all of the three parameters and in what order they are updated are both immaterial, as the content of those three parameters is expressly overwritten by the result of the calling program once control has passed back to it from "Some-Prog".

In summary, for such cases as these, the content of these data items is not likely to reflect a value consistent with expectations associated with their intended purpose; the content instead reflects the fact that their storage area has been overwritten by a value associated with an entirely different purpose.

The user is cautioned that unexpected results may occur when the user fails to ensure that operands either explicitly or implicitly associated with a given statement do not overwrite each other during the course of execution of that statement, and that such storage is not shared with other data items that may unexpectedly change the result of execution of other statements.

D.6 Compilation group and run unit organization and communication

Complete data processing problems are frequently solved by developing a set of logically coordinated compilation units, which may be compiled separately or stacked together for compilation. At some time prior to runtime the output from the compiler may be assembled into a complete problem solution. The organization of COBOL compilation groups and run units supports this approach of dividing large problem solutions into small, more manageable, portions that may be programmed and validated independently.

D.6.1 Compilation group and run unit organization

There are two levels in a COBOL environment. These are the source level and the runtime level.

D.6.1.1 Source level organization

At the source level, the most inclusive unit is a compilation group, which is everything that is submitted to a compiler at one time. A compilation group contains a series of source units.

The types of source units are class definitions, factory definitions, function definitions, function prototypes, interface definitions, method definitions, object definitions, program definitions, and program prototypes. A source unit is a set of COBOL statements as specified in this document and consists of an identification division followed optionally by an environment division and/or a data division and/or a procedure division.

A source element is a term used to refer to a source unit excluding any nested source units.

A source unit that itself is not contained within another source unit is called a compilation unit. A compilation unit may be converted by a compiler into a runtime module that either alone, or together with other runtime modules, is capable of being executed. In general, a source unit that is contained within another unit is not converted by a compiler into a runtime module, since the specifications in this document explicitly permit a contained source unit to reference data in a containing source unit.

The procedure division of a function, method, or program is organized into a sequence of procedures of two types. Declarative procedures, normally termed declaratives, are procedures that will be executed only when special conditions occur at runtime. Nondeclarative procedures are procedures that will be executed according to the normal flow of control. Declaratives may contain nondeclarative procedures but these will be executed only during the execution of the declarative that contains them. Nondeclarative procedures may contain other nondeclarative procedures but shall not contain a declarative. Neither declaratives nor nondeclarative procedures may contain programs. In other words, in COBOL the terms "procedure" and "program" are not synonyms.

Further discussion of classes, factories, interfaces, methods, and objects can be found in D.18, Object oriented concepts.

Figure C.2, Compilation group sample structure example, illustrates the structure of a compilation group. Note that all source elements are source units as well. This is only partially indicated in the example.

```
*> Compilation group start
*> Compilation unit F-1 start
*> Source unit F-1 start
FUNCTION-ID F-1.          *> Source element start
...
END FUNCTION F-1.         *> Source element end
*> Source unit F-1 end
*> Compilation unit F-1 end

*> Compilation unit P-1 start
*> Source unit P-1 start
PROGRAM-ID. P-1.
...
    PROGRAM-ID. P-1-1.    *> Source element start
    ...
    END PROGRAM P-1-1.    *> Source element end

*> Source unit P-1-2 start
    PROGRAM-ID. P-1-2.
    ...
*> Source element P-1-2-1 start
        PROGRAM-ID. P-1-2-1.      *> Source element start
        ...
        END PROGRAM P-1-2-1.      *> Source element end
        PROGRAM-ID. P-1-2-2.      *> Source element start
         ...
        END PROGRAM P-1-2-2.      *> Source element end
    END PROGRAM P-1-2.
*> Source unit P-1-2 end
END PROGRAM P-1.
*> Source unit P-1 end
*> Compilation unit P-1 end

*> Compilation unit C-1 start
*> Source unit C-1 start
CLASS-ID. C-1.
...
*> Source unit start
    FACTORY.
    ...
        METHOD-ID. OM-1.   *> Source element start
        ...
        END METHOD OM-1.   *> Source element end
    END FACTORY.
*> Source unit end
*> Source unit start
    OBJECT.
    ...
        METHOD-ID. IM-1.   *> Source element start
        ...
        END-METHOD IM-1.   *> Source element end
        METHOD-ID. IM-2.   *> Source element start
        ...
        END-METHOD IM-2.   *> Source element end
    END OBJECT.
*> Source unit end
END-CLASS C-1.
*> Source unit C-1 end
*> Compilation unit C-1 end
*> Compilation group end
```

Figure C.2 — Compilation group sample structure example

Figure C.3, Compilation group and run unit structures, shows schematically, in an example, the relationships between the components of a compilation group and their corresponding runtime entities. Note that runtime modules resulting from compilation units of the same compilation group need not be part of the same run unit, and runtime modules in the same run unit need not result from compilation units of the same compilation group.

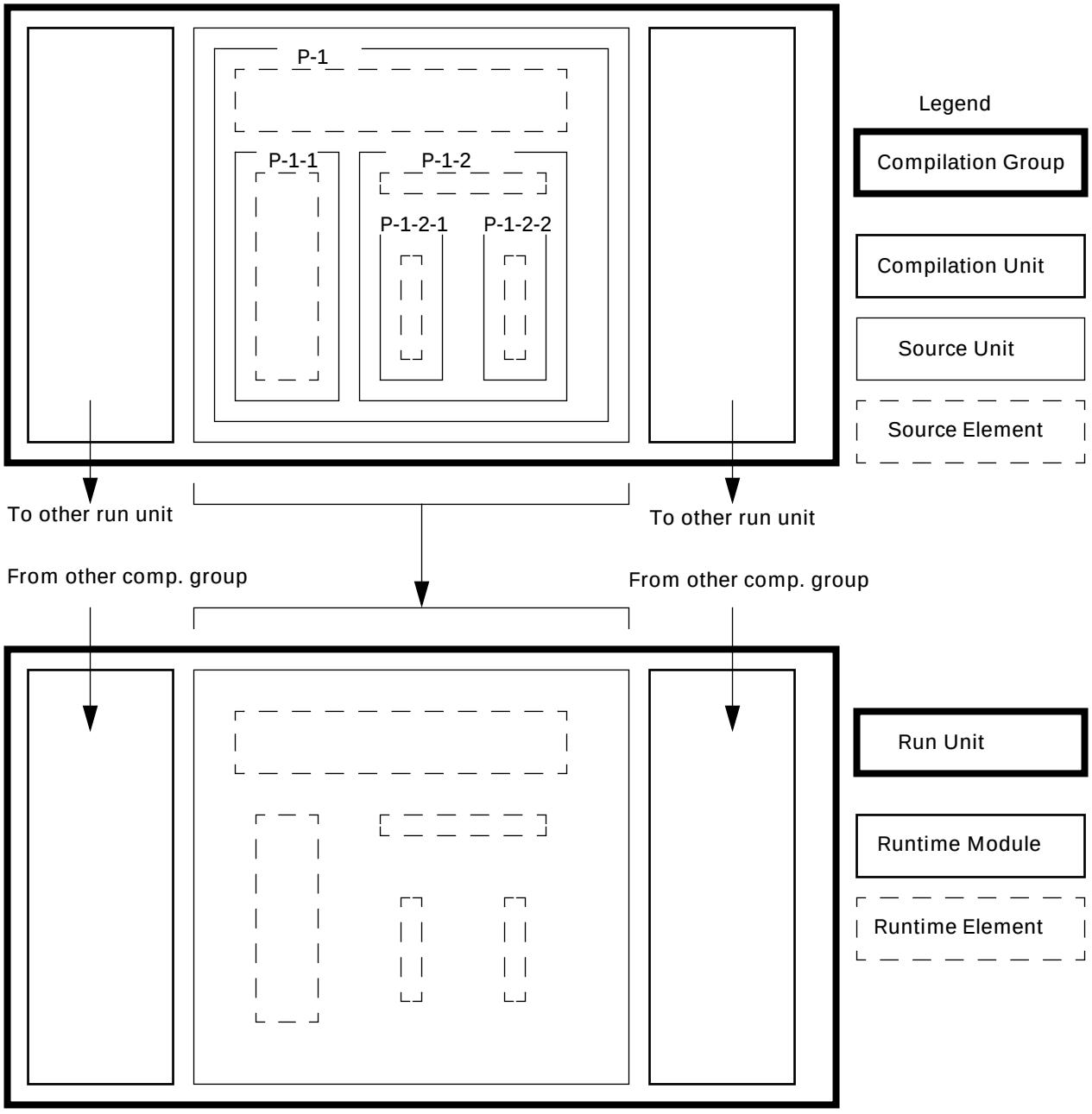


Figure C.3 — Compilation group and run unit structures

D.6.1.2 Runtime level organization

At runtime level, the most inclusive organizational unit is the run unit. A run unit contains one or more runtime modules, as well as other resources needed for the execution of the run unit.

A runtime module results from compiling a compilation unit. Each runtime module contains one or more runtime elements, as well as other resources needed for the execution of those runtime elements.

A runtime element results from the compilation of a function, method, or program.

D.6.1.3 Example

The following compilation group is an example of some of these concepts

```
FUNCTION-ID.  factorial.
DATA DIVISION.
LINKAGE SECTION.
01  parm1 BINARY-LONG.
01  fact  SAME AS parm1.
PROCEDURE DIVISION USING parm1 RETURNING fact.
   IF parm1 = 0
      COMPUTE fact = 1
   ELSE
      COMPUTE fact = parm1 * factorial (parm1 - 1)
   END-IF
   EXIT FUNCTION.
END FUNCTION factorial.

PROGRAM-ID.  program-1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    FUNCTION factorial.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  i  BINARY-LONG.
PROCEDURE DIVISION.
   COMPUTE i = factorial (10)
   ...
END PROGRAM program-1.

PROGRAM-ID.  program-2.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    FUNCTION factorial.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  i  BINARY-LONG.
01  global-item PIC X(30) GLOBAL VALUE "The factorial is: "
PROCEDURE DIVISION.
   COMPUTE i = factorial (11)
   CALL display-it USING i
   ...

   PROGRAM-ID.  display-it.
   LINKAGE SECTION.
   01  n  BINARY-LONG.
   PROCEDURE DIVISION USING n.
      DISPLAY global-item, n
   END PROGRAM display-it.

END PROGRAM program-2.
```

This compilation group consists of three compilation units: the function factorial, the program program-1, and the program program-2. Each compilation unit is also a source unit. The source unit program-2 contains another

source unit, the program display-it. Each of the four source units is also a source element. The difference between the source elements and the source units is that the source element program-2 does not include the nested program.

From each of the 3 compilation units, a separate runtime module is created by the compiler. Typically, a linkage editor is used to combine runtime modules, from one or multiple compilations, into a run unit.

The runtime module for the function factorial contains one runtime element, the function factorial. The runtime module for the program program-1 contains one runtime element. The runtime module for program-2 contains two runtime elements, one for program-2 and one for display-it.

This example illustrates how the user-defined function factorial can override the intrinsic function factorial. The compiler would use the factorial intrinsic function if FUNCTION factorial INTRINSIC were specified in the REPOSITORY paragraph. However, since the INTRINSIC phrase is not specified in the example, the user-defined function is the function that is activated.

D.6.2 Recursive and initial programs

Early COBOL programs always had data in its last-used state and did not allow calling a program when the program was active. Initial and recursive programs allow data to be initialized on every invocation and recursive programs allow programs to be called when they are active. Functions and methods are always recursive.

An initial program is one in which working-storage section internal data items and internal file connectors are set to their initial state whenever the program is called. Data items and file connectors declared as external are left in their last-used state. When the program exits (with EXIT PROGRAM or GOBACK), all programs that are contained in it are canceled (as if CANCEL were executed for each one) and any internal file connectors are closed. This type of program is most often used when it is not necessary to retain data from one invocation to the next. To retain specific data, external data items or file connectors can be used. In many implementations of initial programs, working-storage is allocated "on the stack", which can conserve space because it goes away when the program terminates.

A recursive function, method, or program is one that can be called when it is already active. For example, program A can call B that can in turn call A again. Or, A can call A as illustrated in the example in D.6.1.3 where the function factorial calls itself. Typically, in a recursive program a local-storage section is specified for data that is initialized on each invocation of the program (this data is called automatic data). Working-storage data is static and is therefore in its last-used state on every invocation. The programmer should be aware of this because it can cause unexpected results. For example, if you had a counter "xyz" in the working-storage in one recursion of a program and you added to it, when you got back to another recursion it would be one more than it was before. You might or might not want this to happen.

D.6.3 Accessing data and files

Some data items have associated with them a storage concept determining where data item values and other attributes of data items are represented with respect to the runtime elements of a run unit. Likewise, file connectors have associated with them a storage concept determining where information concerning the positioning and status of a file and other attributes of file processing are represented with respect to the runtime elements of a run unit.

D.6.3.1 Names

A data-name names a data item. A file-name names a file connector. These names are classified as either global or local.

A global name is established by coding a GLOBAL clause in a file description entry or a data description entry. The global name can then be used in the declaring source element and in any source element that is contained, directly or indirectly, within the declaring source element. All uses of a given global name reference the file description entry or data description entry described with that name and the GLOBAL clause in the declaring source element.

A local name, however, may be used only to refer to the item with which it is associated from within the program in which the local name is declared. Some names are always global; other names are always local; and some other names are either local or global depending upon specifications in the program in which the names are declared.

A data-name, file-name, or report-name described using a GLOBAL clause is a global name. All data-names subordinate to a global name are global names. All condition-names associated with a global name are global names. However, specific rules sometimes prohibit specification of the GLOBAL clause for certain data description, file description, or record description entries.

A file-name is global if the GLOBAL clause is specified in the file description entry for that file-name.

If a data-name, a file-name, or a condition-name declared in a data description entry is not global, the name is local.

D.6.3.2 Items

Accessible data items usually require that certain representations of data be stored. File connectors usually require that certain information concerning files be stored. The storage associated with a data item or a file connector may be external or internal to the runtime element in which the item is declared.

D.6.3.2.1 Item types

D.6.3.2.1.1 Working-storage records

Working-storage records are allocations of sufficient storage to satisfy the record description entries in that section. Each record description entry declares a different item. Renaming and redefining do not declare new items; they provide alternate groupings or descriptions for items that have already been declared.

D.6.3.2.1.2 File connectors

File connectors are storage areas that contain information about a file and are used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

D.6.3.2.1.3 Record areas for files

No particular record description entry in the file section is considered to declare the storage area for the record. Rather, the storage area is the maximum required to satisfy associated record description entries. These entries may describe fixed- or variable-length records. In this presentation, record description entries are said to be associated in two cases. First, when record description entries are subordinate to the same file description entry, they are always associated. Second, when record description entries are subordinate to different file description entries and these file description entries are referenced in the same SAME RECORD AREA clause, they are associated. All associated record description entries are redefinitions of the same storage area.

D.6.3.2.1.4 Screen records

A screen record is a conceptual entity that groups together one or more screen description entries. Each screen description entry declares a different screen item. Screen records are not allocations of storage.

A screen record provides the two-dimensional framework within which screen items may be positioned relative to each other and relative to the first line and first column of the terminal display. Positioning of screen items within the screen record is unaffected by how the screen record is referenced; it is the same whether the whole screen record is displayed or only a portion.

D.6.3.2.1.5 Other items

Examples of other items declared in COBOL functions and programs are: report description entries and control information associated with the linkage and report sections.

D.6.3.2.2 Item attributes

A data item or file connector may be an external item or internal item. The storage associated with an external item is associated with the run unit rather than with any particular runtime element within the run unit. An external item may be referenced by any runtime element that describes the item. References to an external item from different runtime elements using separate descriptions of the item are always to the same item. A literal may be used to specify the external name in environments where names are case-sensitive or do not conform to the rules for COBOL name formation.

An item is internal if the storage associated with that item is associated with the runtime element in which it is specified. Internal items may be automatic, initial, or static. If it is automatic or initial, it is initialized every time the runtime element in which it is specified is activated. If it is static, it retains its contents between activations unless a CANCEL statement causes it to be reinitialized. The determination of whether or not an item is automatic or static is made by the section in which the item is described. A data item is initial when it is defined in the working-storage or file section of an initial program.

External and internal items may have either global or local names.

D.6.3.2.2.1 Working-storage records

A record described in the working-storage section is given the external attribute by the presence of the EXTERNAL clause in its data description entry. Any data item described by a data description entry subordinate to an entry describing an external record also attains the external attribute. If a record or data item does not have the external attribute, it is part of the internal data of the runtime element in which it is described.

D.6.3.2.2.2 File connectors

A file connector is given the external attribute by the presence of the EXTERNAL clause in the associated file description entry. If the file connector does not have the external attribute, it is internal to the runtime element in which the associated file-name is described.

D.6.3.2.2.3 Other items

Records, subordinate data items, and various associated control information described in the linkage, report, and screen sections of a runtime element are always considered to be internal to the runtime element describing that data. Special considerations apply to data described in the linkage section whereby an association is made between the records described and other data items accessible to other runtime elements. (See D.6.5.2, Passing arguments.)

D.6.3.3 Name resolution

Certain conventions apply when programs contained within other programs assign the same names to data items, conditions, and file connectors. Consider the situation when program A contains program B which itself contains program C; further, programs A and B, but not program C, contain data division entries for a condition-name, data-name, or a file-name named DUPLICATE-NAME.

1) If either DUPLICATE-NAME references an internal item, two different though identically named items exist. If both DUPLICATE-NAMEs reference an external item, only one item exists.

2) Program A's reference to DUPLICATE-NAME is always to the item that it declares. Program B's reference to DUPLICATE-NAME is always to the item that it declares.

3) If DUPLICATE-NAME is a local name in both programs A and B, program C cannot refer to that name.

4) If DUPLICATE-NAME in program B is a global name, program C may access the item referenced by the name in program B, regardless of whether or not DUPLICATE-NAME is a global name in program A.

5) If DUPLICATE-NAME in program A is a global name but in program B it is a local name, program C's reference to DUPLICATE-NAME is to the item referenced by the name declared in program A.

D.6.4 Program attributes

COBOL programs that form part of a run unit may possess none, one, or more of the following attributes: common, initial, and recursive.

D.6.4.1 Common programs

A common program is one that, despite being directly contained within another program, may be called by any program directly or indirectly contained in that other program. The common attribute is attained by specifying the COMMON phrase in a program's identification division. The COMMON phrase facilitates the writing of nested programs that are to be used by all the programs contained within a program.

D.6.4.2 Initial programs

An initial program is one whose program state is initialized when the program is called. Thus, whenever an initial program is called, its program state is the same as when the program was first called in that run unit. The initial attribute is attained by specifying the INITIAL phrase in the program's identification division.

D.6.4.3 Recursive programs

A recursive program is one that may be called when it is active. It may call itself or a program that it calls may call it before returning control. The recursive attribute is attained by specifying the RECURSIVE phrase in the program's identification division.

D.6.5 Inter-program communication

When the complete solution to a data processing problem is subdivided into more than one runtime element, the constituent runtime elements will be able to communicate with each other. This communication may take four forms: the transfer of control, the passing of parameters, the reference to common data, or the reference to common files. These four inter-program communication forms are provided both when the communicating runtime elements are separately compiled and when one of the communicating programs is contained within the other program. The precise mechanisms provided in the last two cases differ from those in the first two cases; for example, a program contained within another program may reference any data-name or file-name possessing a global name in the containing program. (See D.6.3.1, Names.)

D.6.5.1 Transfer of control

Control is transferred to a program with the CALL statement. Control is transferred to a function by referencing the function-identifier. Control is transferred to a method with either the INVOKE statement or with inline method invocation. The runtime element from which control is transferred is called the activating runtime element; the runtime element to which control is transferred is called the activated runtime element.

D.6.5.1.1 Transfer of control to a program

The CALL statement provides the means whereby control may be transferred to a program within a run unit. A called program may itself contain statements that transfer control to other runtime elements, for example other CALL statements.

When control is transferred to a called program, execution proceeds from statement to statement beginning with the first nondeclarative statement of the called program. If control reaches a STOP statement, this signals the logical end of the run unit. If control reaches an EXIT PROGRAM statement, this signals the logical end of the called program only, and control then reverts to the next executable statement following the CALL statement in the calling runtime element. If control reaches a GOBACK statement and the program has been called, controls continues as for the EXIT PROGRAM statement; otherwise, control continues as for the STOP statement. Thus the EXIT PROGRAM statement terminates only the execution of the program in which it occurs, the STOP statement

terminates the execution of a run unit, and the GOBACK statement returns from whence it came, whether it was another program or the operating system.

The CALL statement may be used to call a program that is not written in COBOL, but the return mechanism and inter-program data communication are not specified in this document. A COBOL program may also be called from a runtime element that is not written in COBOL, but the calling mechanism and inter-program data communication are not specified in this document. In both the above cases, only those parts of the parameter passing mechanism that apply to the COBOL program are specified in this document.

D.6.5.1.1.1 Names of programs

In order to call a program, a CALL statement identifies the program's name or its address. The names assigned to programs that directly or indirectly are contained within another program shall be unique within that other program.

The names assigned to each of the outermost programs that constitute a run unit shall be unique within that run unit. A literal may be used in the REPOSITORY paragraph to specify the name known externally in environments where names are case-sensitive or contain characters that are not allowed in COBOL names.

D.6.5.1.1.2 Scope of the CALL statement

Any runtime element may call an outermost program in the run unit, including itself if it is a recursive program.

In addition, programs may call the following:

1) A program may call any program possessing the common attribute that is directly contained within a program that itself directly or indirectly contains the calling program, unless the calling program is itself contained within the program possessing the common attribute and that program is not a recursive program.

2) A program may call a program that neither possesses the common attribute nor is separately compiled if, and only if, one of the following is true:

    a) the called program is directly contained within the calling program, or

    b) the called program is a recursive program that directly or indirectly contains the calling program.

The calling program may possess any or none of the program attributes, it may either be separately compiled or not, and it may either be contained within programs or contain other programs.

D.6.5.1.1.3 Scope of names of programs

Certain conventions apply when, within an outermost program, a name identical to that specified for another outermost program in the run unit is specified for a contained program.

Consider the situation when program A contains program B and program DUPLICATE-NAME, program B contains program BB, and program DUPLICATE-NAME contains program DD.

The name DUPLICATE-NAME has also been specified for an outermost program.

1) If program A, but not any of the programs it contains, calls program DUPLICATE-NAME, the program activated is the one contained within program A.

2) If either program B or program BB calls program DUPLICATE-NAME then:

    a) If the program DUPLICATE-NAME contained within program A possesses the common attribute, it is called.

b) If the program DUPLICATE-NAME contained within program A does not possess the common attribute, the outermost program is called.

3) If either program DD or program DUPLICATE-NAME contained within program A calls program DUPLICATE-NAME, the program called is the outermost program, except when DUPLICATE-NAME is a recursive program.

4) If any other outermost program in the run unit or any other program contained within such a program calls the program DUPLICATE-NAME, the program called is the outermost program named DUPLICATE-NAME.

D.6.5.1.2 Transfer of control to a function

Control is transferred to a function when it is referenced as a function-identifier.

When control is transferred to a function, execution proceeds from statement to statement beginning with the first nondeclarative statement of the activated function. If control reaches a STOP statement, this signals the logical end of the run unit. If control reaches an EXIT FUNCTION or GOBACK statement, this signals the logical end of the function only and control reverts to the statement that activated the function. Thus the EXIT FUNCTION or GOBACK statement terminates only the execution of the function in which it occurs, while the STOP statement terminates the execution of a run unit.

A function may be activated that is not written in COBOL, but the return mechanism and inter-program data communication are not specified in this document. A COBOL function may also be activated from an element that is not written in COBOL, but the calling mechanism and inter-program data communication are not specified in this document. In both of these cases, only those parts of the parameter passing mechanism that apply to COBOL are specified in this document.

D.6.5.1.2.1 Names of functions

In order to activate a function, a function-identifier identifies the function's name. A literal may be used in the REPOSITORY paragraph to specify the name known externally in environments where names are case-sensitive or contain characters that are not allowed in COBOL names.

The names assigned to each of the functions within a run unit shall be unique.

D.6.5.1.2.2 Scope of a function-identifier

Any runtime element may activate any function in a run unit. All functions are recursive. A function may activate itself and may be activated while it is active.

D.6.5.1.3 Transfer of control to a method

Control is transferred to a method with the INVOKE statement or with inline method invocation.

When control is transferred to a method, execution proceeds from statement to statement beginning with the first nondeclarative statement of the activated method. If control reaches a STOP statement, this signals the logical end of the run unit. If control reaches an EXIT METHOD or GOBACK statement, this signals the logical end of the method only and control reverts to the statement that activated the method. Thus the EXIT METHOD or GOBACK statement terminates only the execution of the method in which it occurs, while the STOP RUN statement terminates the execution of a run unit.

A method may be activated that is not written in COBOL, but the return mechanism and inter-program data communication are not specified in this document. A COBOL method may also be activated from an element that is not written in COBOL, but the activating mechanism and inter-program data communication are not specified in this document. In both of these cases, only those parts of the parameter passing mechanism that apply to COBOL are specified in this document.

All methods are recursive. A method may invoke itself and may be invoked while it is active.

More details about invoking methods are given in D.18, Object oriented concepts.

D.6.5.2 Passing arguments

A function, program, or method is activated in order to perform, on behalf of the activating runtime element, some defined part of the solution of a data processing problem. In many cases it is necessary for the activating runtime element to make certain data values available to the activated runtime element, which are required for its part of the problem solution. One method for ensuring the availability of these data values is by passing arguments, as is described in this paragraph. Another method is to share the data. (See D.6.5.3, Sharing data.) The data values passed as arguments also may identify some data to be shared; therefore the two methods are not mutually independent.

D.6.5.2.1 Identifying arguments

A data item passed as an argument by a runtime element activating another runtime element is accessible in the activated runtime element. The activated runtime element contains a description of each of these arguments, called formal parameter, in the linkage section of the activated runtime element.

In a source element describing the activating runtime element, the values of the arguments to be passed are identified by listing them in form of identifiers, literals, or arithmetic or boolean expressions in the CALL statement, INVOKE statement, inline method invocation, or function identifier, as applicable. In a source element describing the activated runtime element, the expected formal parameters are identified by listing them in that source element's procedure division header. These lists establish, on a positional basis at runtime, the correspondence between the values as they are known to each source element; that is, the first argument on the list of the activating source element corresponds to the first formal parameter on the list of the activated source element, the second argument to the second formal parameter, etc. For example, a program to be called may include:

```
PROGRAM-ID.  EXAMPLE.
PROCEDURE DIVISION USING NUM, PCODE, COST.
```

and may be called by executing:

```
CALL "EXAMPLE" USING NBR, PTYPE, PRICE.
```

thereby establishing the following correspondence:

Called Program (Example)	Calling runtime element
NUM	NBR
PCODE	PTYPE
COST	PRICE

Only the positions of the data-names are significant, not the names themselves.

D.6.5.2.2 Argument passing mechanisms

There are three mechanisms for passing arguments: "by reference", "by content", or – when a prototype is available for the activated element (see below) – "by value".

If the "by reference" or "by content" mechanism is specified (or implied), the location of an argument or of a copy of it is made available to the activated runtime element. If the "by value" mechanism is specified, a value, rather than a location, is made available to the activated runtime element.

If an argument is passed by reference, the activated runtime element is allowed to access and modify the value of the data item referenced by an argument, except for the address-identifier.

If an argument is passed by content or by value, the activated runtime element may access, but not modify, a data item in the activating runtime element. If a prototype is available, the value of the argument is evaluated and converted to the format of the corresponding formal parameter. If no prototype is available, a copy of the argument is presented to the activated runtime element as if it had been passed by reference. In either case, the value passed

may be changed by the activated runtime element during the course of its execution, but the value of the argument in the activating runtime element is not modified.

Thus an argument passed by reference may be used by an activated runtime element to return a result to the activating runtime element, whereas an argument passed by content or by value cannot be so used.

In the source element describing the activated runtime element, the list of formal parameters specified in the procedure division header describes the mechanism for receiving the argument. A formal parameter for which BY REFERENCE is specified in this list may receive an argument that is passed either by reference or by content. A formal parameter for which BY VALUE is specified may receive only an argument that is passed by value.

D.6.5.2.3 Passing addresses

Passing addresses is a special case, because unlike other identifiers that are not defined data items, address-identifiers may be passed in the non-prototype formats of the CALL statements, and with all three passing mechanisms. Note, however, that the address-identifier is not a valid receiving operand; therefore, it will never be updated even when passed by reference. It behaves like being passed by content.

D.6.5.2.4 Returning items

A returning item is required for function calls and inline method invocation; it is optional for program calls and method invocation using the INVOKE statement. Just like arguments, the returning item is allocated in the activating runtime element. The source element describing the activated runtime element contains only a description of the data being returned in its linkage section, analogous to the formal parameters.

In the source element describing the activated element, the returning item is identified by the RETURNING phrase of its procedure division header, which references its description in the linkage section. In the source element describing the activating element, it is either identified by the RETURNING phrase of the CALL or INVOKE statement, or it becomes the result of the evaluation of the function-identifier or the inline method invocation.

D.6.5.2.5 Prototypes

Program prototypes, function prototypes, and method prototypes specify the types of arguments and returning items that are expected by a program, function, or method. With this information the system can check that the arguments that are passed and the returning item match the formal parameters describing the data expected by an activated runtime element. The procedure division header shows the number of parameters that are expected and returned. The linkage section gives a description of the parameters and of the returned value. The prototype may also specify an entry-convention that is different from the default COBOL entry-convention on a system, in order to facilitate communication with functions, methods, and programs written in other programming languages.

For programs, the CALL statement has formats for which a prototype is not used and a format in which the prototype is used. Function calls and method invocation always use a prototype.

Use of the by content or by value mechanism along with a prototype allows the system to convert arguments during the activating process into the format expected by the activated function, method, or program. It also allows passing identifiers that do not reference data items and passing arithmetic and boolean expressions.

A prototype may specify that a parameter can be omitted when the function, method or program is referenced. The OMITTED test allows a runtime element to determine whether a parameter is present.

Prototype information may be available in the same compilation group or in an external repository. Prototype information for nested programs, identified by the NESTED phrase of the CALL statement, is extracted from the actual program to be called. Specification of a function-prototype-name or a program-prototype-name in the REPOSITORY paragraph causes the system to find the information not specified in the compilation group in the external repository for prototypes. For methods, the prototype is provided through the class or interface in which the method is defined.

D.6.5.2.6 Defaults when no prototype is used

When no prototype is used (which can only occur with a program call), the passing mechanisms for the arguments are completely defined in the CALL statement. When neither BY REFERENCE nor BY CONTENT is specified for an argument, BY REFERENCE is assumed. When either is specified, this also applies to subsequent arguments as long as not superseded by another specification of either of these phrases; that is, the specification is "transitive".

D.6.5.2.7 Defaults when a prototype is used

When a prototype is used, the passing mechanisms are basically determined by the prototype specification. When none of the BY REFERENCE, BY CONTENT, or BY VALUE phrases are specified for an argument, the passing mechanism is determined as follows:

— When BY VALUE is specified in the prototype, BY VALUE is assumed.

— When BY REFERENCE is specified in the prototype and the argument is an item that is a valid receiving operand (other than an object property), BY REFERENCE is assumed.

— When BY REFERENCE is specified in the prototype and the argument is an item that is not a valid receiving operand, BY CONTENT is assumed.

The CALL and INVOKE statements allow override of the default outlined above by specifying the BY CONTENT phrase for an argument that is a valid receiving operand. In this case BY CONTENT is used rather than BY REFERENCE.

Note that specifying BY REFERENCE or BY VALUE in an INVOKE statement or the prototype format of a CALL statement has no effect other than consistency checking and documentation.

Also note that there is no transitivity for the passing mechanism when a prototype is used.

D.6.5.3 Sharing data

Runtime elements in a run unit may reference common data in the following circumstances:

1) The data content of an external data record may be referenced from any runtime element in which that record is described as external. There is one instance of that record associated with the run unit.

2) If a program is contained within another program, both programs may refer to data possessing the global attribute declared either in the containing program or in any program that directly or indirectly contains the containing program. (See D.6.3.1, Names.)

3) The mechanism whereby a parameter value is passed by reference from an activating runtime element to an activated runtime element establishes a common data item; the activated element, which may use a different identifier, may refer to that data item in the activating element.

4) The mechanism whereby a value is returned from a function, method, or program establishes a common data item, the returning data item. The activated runtime element, which may use a different identifier, may return a value in the returning data item in the activating runtime element.

D.6.5.4 Sharing files

Files can be shared across run units, across runtime elements, or by different file connectors within a given runtime element as described in D.2.3, File sharing and record locking.

Two runtime elements in a run unit may reference common file connectors in the following circumstances:

1) An external file connector may be referenced from any runtime element that describes that file connector as external. There is one instance of that file connector associated with the run unit.

2) If a program is contained within another program, both programs may refer to a common file connector by referring to an associated global file-name declared either in the containing program or in any program that directly or indirectly contains the containing program. (See D.6.3.1, Names.)

D.7 Intrinsic function facility

The intrinsic function facility provides a means of returning a value that is derived from a specific algorithm or from an evaluation of one or more arguments provided to the function. The resulting value is considered to be a temporary data item, and can be a character string, a bit string, or a numeric value. Functions returning numeric values can be used in arithmetic expressions just like any numeric data item.

The user invokes an intrinsic function by specifying the word FUNCTION followed by the name of the function optionally followed by arguments in parentheses. For example:

```
MOVE FUNCTION MAX (1, a) TO b
```

Intrinsic-function-names are not reserved words and the word FUNCTION is used so they do not have to be reserved. However, the user can specify that one or more intrinsic-function-names are to be used without the word FUNCTION by using the intrinsic format of the function-specifier in the REPOSITORY paragraph. For example, if you do not want to use the word FUNCTION before any intrinsic-function-names in a source unit, you can specify the following repository paragraph:

```
REPOSITORY.
    FUNCTION ALL INTRINSIC.
```

and all of the intrinsic function names can be referenced without being preceded by the word FUNCTION. However, in this example none of the intrinsic function names could then be specified as user-defined words for the scope of that REPOSITORY paragraph. If you want to omit the word FUNCTION before one or more particular intrinsic function names, individual intrinsic function names can be specified, for example:

```
REPOSITORY.
    FUNCTION CURRENT-DATE, DAY-OF-INTEGER INTRINSIC.
```

Intrinsic functions differ from user-defined functions in that they are intrinsic to the language as opposed to being written by the user. The COBOL definitions of the intrinsic functions give the allowed arguments, their categories, their ranges, and the method that is used to determine the returned value.

Some functions have no arguments because they provide a known quantity. An example is the CURRENT-DATE function, which provides information about the current date, time and difference from UTC. For example, you can specify

```
MOVE FUNCTION CURRENT-DATE TO the-date-info
```

or, if you are interested only in YYYYMMDD you can specify

```
MOVE FUNCTION CURRENT-DATE (1: 8) TO the-date
```

and you will get a date of the form yyyymmdd. Notice that you can reference modify any function that returns a string. An example of a function that may have an argument or not is the RANDOM function. It can have an argument to specify a seed value or no argument. In both cases it returns a pseudo-random number.

Some functions that allow a variable number of arguments can be specified with a table as an argument, which means that the entire table is the series of arguments. An example is the MAX function where one may specify

```
MOVE FUNCTION MAX (a-table (ALL)) TO a-variable
```

If you are interested only in the first part of a return that is a string (assume a-table is a table of PIC X(10) items) you may specify

```
MOVE FUNCTION MAX (a-table (ALL)) (1: nbr-chars) TO a-variable
```

These will select the maximum value from all a-table items. You could use FUNCTION ORD-MAX (a-table (ALL)) to figure out which element of the table was the largest.

There are exception-names associated with functions that can be used to determine if the arguments were correct. An example is:

```
>>TURN EC-ARGUMENT CHECKING ON
COMPUTE day-difference = FUNCTION INTEGER-OF-DATE (first-date) -
  FUNCTION INTEGER-OF-DATE (second-date)

IF FUNCTION EXCEPTION-STATUS = "EC-ARGUMENT"
   ...     *> here one would attempt to figure out what was wrong.
```

There are functions to check the arguments for the date and numeric value functions such as TEST-DATE-YYYYMMDD and TEST-NUMVAL.

D.8 Types

A type is a template that contains all the characteristics of a data item and its subordinates. A type is declared and named by specifying the TYPEDEF clause. The essential characteristics of a type, which is identified by its type-name, are the relative positions and lengths of the elementary items defined in the type declaration, and the BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, SYNCHRONIZED, and USAGE clauses specified for each of these elementary items.

There are three different kinds of typed items:

— weakly-typed elementary items
— weakly-typed group items
— strongly-typed group items

Note that elementary items cannot be strongly-typed.

A type – whether weak or strong – defines a certain data structure, which has a specific name, and, when used, is used with exactly that defined structure. This ensures that this structure is not affected by things like different alignment, or by changing the data representation by means of a USAGE specification on a higher group level.

D.8.1 Weakly-typed items

Weakly-typed items refer to a type declaration that does not include the STRONG phrase and are not subordinate to such a type declaration. Weakly-typed items can be either group items or elementary items.

Other than preserving the data structure as described above, weakly-typed items can essentially be used just like any untyped items. Thus, the TYPEDEF can indeed be regarded as a "shorthand" for a series of data description entries.

The following example illustrates the use of the TYPEDEF clause and the TYPE clause to define and use a type:

```
1   Feature TYPEDEF.                        *> defines a type-name Feature
    2    Feature-name PIC X(15) OCCURS 10.  *> with this description

1   Equipment.
    2    Equipment-id OCCURS 100 TIMES.
         3    Feature-list TYPE Feature.   *> uses type-name Feature
```

This results in a record with the description:

```
1    Equipment.
     2    Equipment-id OCCURS 100 TIMES.
          3    Feature-list
               4    Feature-name PIC X(15) OCCURS 10.
```

The expansion of a type can create a hierarchy deeper than the 49 levels that can be directly coded in a data description entry.

D.8.2 Strongly-typed group items

Strongly-typed group items are described by a type declaration that includes the STRONG phrase or are subordinate to a type declaration with the STRONG phrase. Only group items can be strongly-typed, because strong typing for elementary items would impose a large number of substantial restrictions on such items, which would make their use all but impractical.

The main purpose of strong typing, beyond preserving the data structure defined by the type, is to protect the integrity of the data contents. Thus, proper usage of a strongly-typed group item should never yield "bad data", i. e., data contents that are incompatible with their data description. One important consequence of this is that any explicit or implicit redefinitions of such data items with less restrictive data descriptions are prohibited.

A group item can be thought of as a kind of redefinition, specifically a "redefinition" of its subordinate data items as one item - an alphanumeric group item, a bit group item, or a national group item. This has the implication that operations on strongly-typed groups, or groups containing strongly-typed group items, are restricted to those that don't affect the integrity of the data subordinate to the group. Thus, the only way to use strongly-typed group items as receiving operands is when the sending operand is of the same type, where "same type" is defined as a type declaration with the same name and the same essential characteristics, as described above.

Note that there is no need to impose the same kind of restrictions on elementary items, because operations for elementary items in general do not corrupt the contents of the receiving operands.

Addresses of strongly-typed group items and data-pointers containing such addresses are subject to restrictions as well, as described in D.9, Addresses and pointers.

You should be aware that preserving the integrity of data makes sense only if the data is correct in the first place. The use of the COBOL exception handling facilities to detect any violations of the rules defined for the COBOL language assists a program in ensuring that the format of data is correct.

A strongly-typed group item is either a level 1 group item or a group item subordinate to a type declaration with the STRONG phrase.

The restrictions for strongly-typed group items are summarized as follows:

1) The data description entry of a strongly-typed group item cannot contain a VALUE clause, nor can the item be a conditional variable.

2) Strongly-typed group items and elementary items subordinate to strongly-typed group items cannot be any of the following:

    a) implicitly or explicitly redefined

    b) renamed in whole or in part

    c) reference modified, except for elementary items of category alphabetic, alphanumeric, boolean and national.

3) A strongly-typed group item may be referenced as a receiving operand only in one of the following:

    a) a program, function or method activation as a formal parameter or returning item

    b) an INITIALIZE statement

    c) a MOVE statement

d) a READ statement

e) a RELEASE statement with the FROM phrase

f) a RETURN statement

g) a REWRITE statement with the FROM phrase

h) the data item referenced in the DESTINATION clause of an element of the operand of a VALIDATE statement

i) the subject of a data description entry that contains a VALIDATE-STATUS clause that references the element of the operand of a VALIDATE statement

j) a WRITE statement with the FROM phrase.

4) A strongly-typed group item can be compared only with another strongly-typed group item of the same type.

D.9 Addresses and pointers

Addresses and pointers relate to the computer storage and to addresses used to navigate in this storage. Normally, business applications will have no need for these features. However, the increasing use of any language for all purposes, including system programming, and the need for interoperability with systems like POSIX and languages like C and C++ have made it desirable to add these capabilities to the COBOL language as well.

Note, however, that addresses and pointers should be used with great care and only where required by the application or the system environment, because they can easily lead to a programming style that makes applications hard to read and to maintain.

There are three kinds of addresses, and correspondingly three kinds of pointers:

— data-addresses and data-pointers
— function-addresses and function-pointers
— program-addresses and program-pointers

D.9.1 Data-addresses and data-pointers

A data-address is a conceptual entity identifying the location of a data item. It is referenced by specifying a data-address-identifier. A data-address-identifier cannot be a receiving operand. Note that the ADDRESS OF phrase in the receiving operand of a SET statement is not considered a data-address-identifier, but a syntactical notation for setting the address of a based item to the value specified by the sending operand.

A data-pointer is a data item used to store a data-address.

A data-address can be stored in a data-pointer. The data-address of a based item can be set from either a data-pointer or a data-address-identifier. Data-addresses and data-pointers can be passed to other source elements, and data-pointers can be received from another source element.

D.9.1.1 Restricted data-pointers

A restricted data-pointer may contain only the predefined address NULL or the address of a data item of a specific type.

1) by specifying a data description entry that contains a usage clause of the form USAGE POINTER TO type-name-1.

2) by specifying a data-address-identifier (ADDRESS OF identifier-1, where identifier-1 is a strongly typed group item or another restricted data pointer).

Use of restricted data-pointers provides type safety by precluding the treatment of data of one type as data of another type.

This is of special significance for strongly-typed group items. The address of a strongly-typed group item is considered a restricted data-pointer. Therefore, it can only be assigned to a data-pointer restricted to the same type, and in turn it can only be used to address a based item of the same type. Conversely, the address of a strongly-typed based item can only be set to the address of a data item of the same type. This way, the existing restrictions for enforcing the integrity of strongly-typed group items cannot be circumvented by the use of addresses, pointers, and based items.

D.9.1.2 Examples

Consider the following program prototype for a program that returns a pointer to a record.

```
Program-id. Get-next-record is prototype. *> returns the address of a record
Data division.
Linkage section.
01 ptr1 usage pointer.
Procedure division returning ptr1.
End program get-next-record.
```

Suppose a client program has the following REPOSITORY paragraph and data declarations:

```
Repository.
   Program Get-next-record.
...
01 p usage pointer.

01  my-wreck based.
    02 name pic x(30).
    02 addr pic x(30).
```

The following procedure division statement calls the program described by the prototype:

```
Call get-next-record returning p
```

The data can be accessed via my-wreck because the pointer p contains the location of a record.

```
Set address of my-wreck to p
Move "SAM JONES" to name in my-wreck
```

Consider a second example based on the fact that many Application Program Interfaces (APIs) require a pointer as a parameter. The data division contains the following declarations:

```
01 p2 usage pointer.

01 data-record.  >*  the full record layout is described
   02 ...
```

If you want to pass a pointer to the program process-record, you could code:

```
Set p2 to address of data-record
Call "process-record" using p2
```

Or, you could pass the address of data-record with the following single statement:

```
Call "process-record" using address of data-record
```

D.9.2 Program-addresses, function-addresses, program-pointers and function-pointers

A program-address identifies the location of a program. A function-address identifies the location of a function. A program-address is referenced by specifying a program-address-identifier. A function-address is referenced by specifying a function-address-identifier. Program-address-identifiers and function-address-identifiers cannot be used as receiving operands.

A program pointer is a data item that is used to store a program-address. A function pointer is a data item that is used to store a function-address.

A program-pointer or a program-address can be used to call a program. A function-pointer or function address can be used anywhere in a program that a function-identifier may be used. Program-addresses, function-addresses, program pointers, and function pointers can be passed to other source elements. Program and function pointers can be received from another source element.

D.9.2.1 Restricted program-pointers and function-pointers

A restricted program-pointer may contain only the address of a program with the same "signature" as the program specified in the definition of the program-pointer; that is, any prototype information that exists for the specified program is applicable to the program identified by the address. A restricted program-pointer may only be defined in a type declaration.

A function-pointer may only contain the address of a function with the same "signature" as the function specified in the definition of the function-pointer; that is, all explicit or implicit prototype information that exists for the specified function is applicable to the function identified by the address.

D.10 Boolean support and bit manipulation

The term boolean support encompasses all functionality defined for data items having category boolean and described with any of the usages BIT, DISPLAY, or NATIONAL. Bit manipulation can be accomplished by describing boolean items as usage BIT.

Data items of category boolean, referred to as boolean items, are defined by picture character 1. They can be represented in storage either as bits or as characters, where each bit or character has a value of 0 or 1, where 0 is false or OFF, and 1 is true or ON.

The usage specified in the data description of a boolean item can be BIT, DISPLAY, or NATIONAL. If picture symbol 1 is specified and a usage is not specified, the usage defaults to DISPLAY. Usages DISPLAY and NATIONAL are provided for ease of printing or displaying the value of boolean items. All three representations can be manipulated in the same manner. When storage is not an issue, usages DISPLAY and NATIONAL avoid the inconvenience of converting bits to characters for printing or displaying them.

Boolean values can be specified in literals with an opening separator B", for example B"1110" where the value is expressed in bits, or opening separator BX", for example BX"E" where the value is expressed in hexadecimal notation.

Boolean operators B-AND, B-OR, B-XOR, and B-NOT can be used to perform "and", "or", "exclusive or", and "negate" operations, respectively, on boolean items. Boolean operators are used in boolean expressions; for example:

```
01    My-flag    PIC  1111  USAGE BIT VALUE B"0000".
01    My-flag-2  PIC  1111  USAGE BIT.
...
MOVE B"0011" to My-flag-2  *> Initialize My-flag-2
...
COMPUTE My-flag   =   B-NOT  My-flag-2  *>  set the bits in My-flag to the
                                        *>  reverse of My-flag-2, 1100.
...
COMPUTE My-flag   =   My-flag  B-AND  B"0000" *> turn off all the bits in My-flag.
...
COMPUTE My-flag-2   =   My-flag-2   B-OR   BX"8" *>  set bit 1 ON in My-flag-2, keeping
                                              *>  other bits unchanged.
...
```

Alternatively, the last COMPUTE statement could be replaced by:

```
MOVE  B"1"  to   My-flag-2(1:1)   *>  set bit 1 ON using reference
                                  *>  modification
```

Table C.2, Examples of boolean operations, illustrates the result of boolean operations for each of the boolean operators.

Table C.2 — Examples of boolean operations

Boolean operation	Value of operand	Operator	Value of operand	Result
Conjunction	1100	B-AND	0101	0100
Inclusive disjunction	1100	B-OR	0101	1101
Exclusive disjunction	1100	B-XOR	0101	1001
Negation		B-NOT	1100	0011

Boolean items can be tested in two ways:

1) as a boolean condition, if the length of an item is 1 bit position or 1 character position; for example:

```
 01  Single-bit-item pic 1 usage bit.
 01  Multiple-bit-item  pic 1(7) usage bit.
 ...
 IF  Single-bit-item  THEN  CALL  a-program *>  calls if Single-bit-item is true (1)
```

or, using reference modification; for example:

```
 IF  Multiple-bit-item (4:1) THEN ...
```

2) as a relation condition, testing for equal or not equal; for example:

```
              IF   My-flag-2   EQUAL   B"1000"  THEN CALL a-program
```

Boolean items can be converted to integer with the INTEGER-OF-BOOLEAN intrinsic function, and integer items can be converted to boolean with the BOOLEAN-OF-INTEGER intrinsic function. For example:

```
01  bit-item  PIC 1(8) usage BIT.
01  integer-item   PIC 9(5)  VALUE  544.
01  integer-item-2   PIC 9(3).
...
MOVE  FUNCTION  BOOLEAN-OF-INTEGER ( integer-item ,   6) TO bit-item.
    *>  the function returns the low order 6 bits of the binary representation of the
    *>  integer value, and MOVE stores it in bit-item, padding on the right with 0's

COMPUTE integer-item-2  =  FUNCTION  INTEGER-OF-BOOLEAN (bit-item (1:6) ).
    *>  the function returns the numeric value of the leading 6 bits, 32, and
    *>  COMPUTE puts 032 in integer-item-2
```

The rules of COBOL use the terms "character position" or "boolean position". Because it is important to understand how bits are aligned in storage, the following explanation uses the term "byte", which usually corresponds to one alphanumeric character position.

The alignment of items of usage bit depends on how the data description entry is written. If items of usage bit are redefined, or if they redefine an item with a different usage, correct redefinition could depend on knowing the alignment as well as the characteristics of storage in the processor being used, just as it can for items of other representations. The following examples illustrate bit alignment, making the assumption that the bits of a byte are numbered left to right from 1 to 8, assuming an 8-bit byte.

The simplest case is that of a single elementary data item. The first bit is aligned on a byte boundary and subsequent bits are mapped contiguously, continuing across byte boundaries if necessary. Filler bits are not used to fill up a partially-used byte. The following example illustrates this, where a '1' indicates an assigned bit position.

```
                                  Bit position        Byte # in data item
                                  1 2 3 4 5 6 7 8

77 Item-1 PIC 111 USAGE BIT.      1 1 1               1

 1 Item-2 PIC 1(10) USAGE BIT.    1 1 1 1 1 1 1 1     1
                                  1 1                 2
```

The length of Item-1 is 3 boolean positions; the length of Item-2 is 10 boolean positions. The unused bits are not accessible except with REDEFINES of a level 1 item.

Bits can be defined within an alphanumeric group item, within a strongly-typed group item, or within a group described with the GROUP-USAGE BIT phrase. The GROUP-USAGE BIT phrase defines a bit group, which is treated as an elementary bit data item in COBOL operations, except in operations designed for specific group processing, such as the INITIALIZE statement and MOVE CORRESPONDING.

Within an alphanumeric group item, a bit group item, or a strongly-typed group item, the first bit data item is aligned on a byte boundary and bits are assigned consecutively until

- a non-bit item or non-bit group item is encountered;
- an item is reached that is defined with an ALIGNED clause; or
- the end of the group is reached.

The following example illustrates the generation of implicit filler to align on a byte boundary in an alphanumeric group item:

```
                                        Bit position     Byte # in data item
                                        1 2 3 4 5 6 7 8

01 group-1.                                              1-6
   02  item-1  pic 11  usage BIT.       1 1             1
   02   item-2  pic 1  usage BIT.           1           1
     *> implicit filler ...1                  1 1 1 1 1 1      02 PIC 1(5)
   02  item-3 PIC X(3).                                 2-4
   02  item-4  pic 1 usage BIT.         1               5
     *> implicit filler ...                 1 1 1 1 1 1 5      02 PIC 1(7)
   02  group-2.                                         6
       03  item-5  pic 1 usage BIT.     1               6
       03  item-6  pic 1(4) usage BIT.    1 1 1 1       6
         *> implicit filler ...                   1 1 1 6      02 PIC 1(3)
```

The generated filler is not part of the preceding data, but is part of any groups that contain the item. For example, the filler generated after item-6 is included in group-2 and group-1, but not in item-6. The filler after item-6 is added to make the group end at a byte boundary.

If group-2 were described with the GROUP-USAGE BIT phrase, the alignment would be different, because there is no automatic alignment for a bit group following a bit item:

```
                                        Bit position     Byte # in data item
                                        1 2 3 4 5 6 7 8

01 group-1.                                              1-5
   02  item-1  pic 11 usage BIT.        1 1             1
   02  item-2  pic 1  usage BIT.            1           1
     *> implicit filler ...                   1 1 1 1 1 1      02 PIC 1(5)
   02  item-3 PIC X(3).                                 2-4
   02  item-4  pic 1 usage BIT.         1               5
   02  group-2 GROUP-USAGE BIT.                         5
       03  item-5  pic 1 usage BIT.       1             5
       03  item-6  pic 1(4) usage BIT.      1 1 1 1     5
         *> implicit filler ...                   1 1 5      02 PIC 1(2)
```

The ALIGNED clause can be used to override the default bit alignment; in the following example:

```
                                        Bit position     Byte # in data item
                                        1 2 3 4 5 6 7 8
01 group-1.                                     1-2
   02   item-1  pic 11 usage BIT.       1 1             1
     *> implicit filler ...                 1 1 1 1 1 1  1      02 PIC 1(6)
   02   item-2  pic 1 ALIGNED
         usage BIT.                     1               2
     *> implicit filler ....              1 1 1 1 1 1 1  2      02 PIC 1(7)
```

the use of the ALIGNED clause caused item-2 to be aligned on a byte boundary.

It is necessary that the programmer ensure alignment on a byte boundary for bit strings being used as arguments for CALL, INVOKE, a function reference, or an inline method invocation, and when using the ADDRESS OF identifier for bit strings. These operations require that the bit item be aligned on a byte boundary so that it is directly addressable. The ALIGNED clause can be used to ensure byte boundary alignment.

When filler bits are needed at the end of a record, the level number of the filler will depend on the hierarchical structure of the record. The filler level number is the same as the highest hierarchical level superordinate to the last data item, excluding level 1, or, if there is no such superordinate item, the same as the last data item in the record. This is illustrated by the following two examples:

*> filler at the same level as the last data item in the record

```
1  G1.
   2 G1-a USAGE BIT PIC 1.
   2 G1-b USAGE BIT PIC 11.
*> 2  implicit filler  USAGE BIT PIC 1(5).
```

*> filler at the level of the highest level superordinate item

```
1  G2.
   2  Fld-1 PIC X.
   2  Fld-2 GROUP-USAGE BIT.
      3 G2-A.
         5 B-1  PIC 1 USAGE BIT.
         5 B-2  PIC 1 USAGE BIT.
*> 2  implicit filler  USAGE BIT PIC(6).
```

D.11 Character sets

COBOL has these character set concepts:

1) the COBOL character repertoire,

2) the computer's coded character set, and

3) alphabets.

The COBOL character repertoire defines the characters that are used to write the COBOL words and separators that form a compilation group. The repertoire consists of the characters as abstract entities, independent of their encoding. The implementor maps the COBOL character repertoire to an encoding, such that each character of the repertoire is assigned to one or more bit patterns. The resultant encoding is called a coded character set, or sometimes just character set.

At compile time, comments and the non-hexadecimal format of alphanumeric and national literals may contain any of the characters that the implementor has defined in the coded character set, except any used to end a free-form line. COBOL words and separators are limited to characters in the COBOL character repertoire. This is the significant difference between the COBOL character repertoire and the compile-time computer's coded character set used.

At runtime, data is represented in the storage in the computer's runtime coded character set, which may be the same coded character set used at compile time or may be a different one. If the coded character sets used at compile time and runtime are different, the content of alphanumeric and national literals are translated from the compile-time coded character set to the runtime coded character set.

Data on external media may be represented in the computer's runtime coded character set or may be represented in a different coded character set. When the coded character sets are different, a CODE-SET clause specified in the file control entry causes data to be converted between the two coded character sets on input and output.

Alphabets refer to character sets programmed in the compilation group, or provided by the implementor, or specified by national or international standards. A runtime element may use alphabets for selecting collating sequences or for specifying coded character sets to be used in conversion of data on input or output. The ALPHABET clause in the SPECIAL-NAMES paragraph is used to identify alphabets that may be used in the compilation unit.

D.11.1 Character set representations

The representation of a character set is the encoding used to record characters for processing by the computer or for storage on external media. COBOL supports two types of character set representation — alphanumeric and national, corresponding to usages display and national, respectively. These two representations may be implemented as:

- a single character set logically viewed as separate alphanumeric and national coded character sets, or
- two separate coded character sets, one alphanumeric and one national.

The term "computer's coded character set" refers to either character set or both, depending on context.

COBOL uses the term alphanumeric typically to refer to character sets used in information technology to represent a minimal set of characters, usually 128 or 256. An example of this type of character set is ISO/IEC 646. The term national refers to character sets used to represent very large sets of characters. One national character set is the Universal Multiple-Octet Coded Character Set (UCS) defined in ISO/IEC 10646, which includes the characters used in the written form of most of the languages of the world. However, there are many alphanumeric character sets and many national character sets that may be used with COBOL. The implementor specifies which are supported for a specific COBOL implementation.

For purposes of the COBOL character repertoire used in the syntax of a compilation group, a character is a character regardless of its type of representation. The letter 'A' has the same meaning whether it is represented in an alphanumeric character set or a national character set, just as lowercase 'a' has the same meaning as uppercase 'A'.

D.11.2 Programming to use alphanumeric and national character data

COBOL provides two classes of character data -- alphanumeric and national. Data of class alphanumeric is held in data items described with usage DISPLAY. Data of class national is held in data items described with usage NATIONAL.

Most programmers are already familiar with usage DISPLAY, so these concepts will address usage NATIONAL.

The following categories of data can be represented in usage national:

- national, described with picture symbol N;
- national-edited, described with picture symbols N, B, 0, or /, with at least one N and one other symbol in the picture character-string;
- numeric, described with picture symbols 9, P, S and V, the same as any numeric picture;
- numeric-edited, described with any of the picture symbols that define a fixed-point numeric-edited item (0, 9, V, Z, ...);

— boolean, described with picture symbol 1.

The following illustrates data description entries for defining data items with usage national:

```
01  a-rec usage national.
    02  nat-item         picture   N(10).      *> category national
    02  nat-ed-item      picture   NN/NN/NNNN. *> category national edited
    02  num-item         picture    9(5).      *> category numeric
    02  num-edited-item  picture   +99.99.     *> category numeric-edited
    02  bool-item        picture     1111.     *> category boolean
```

Although the amount of storage needed for each national character may be greater than the amount of storage needed to hold an alphanumeric character, the length of a national item is counted as the number of character positions in the item, and not the number of bytes needed to hold the item. In this example, if national data is represented in 16-bit characters, such as UTF-16, the length of nat-item is 10, although 20 bytes of storage are allocated for the data item.

A literal of class national is identified by the opening literal delimiter N", or for hexadecimal literals, NX"; for example:

NX"02A102A2"

A data item with usage national can be moved, compared, inspected, written, read, displayed, used in computation if its category is numeric, and treated in nearly every respect in every language construct just the same as a data item of usage DISPLAY. One of the differences is that data items of usage national can be moved to data items of usage display only by explicit conversion with the DISPLAY-OF function.

The default class and category of a group item containing data items of usage national is alphanumeric, just as it is for a group item containing other data items. A group can be explicitly given class and category national by coding the GROUP-USAGE NATIONAL phrase at the group level of a data description entry when all subordinate data items in the group, and in any contained groups, are of class and category national. For example:

```
01  Group-1  GROUP-USAGE NATIONAL.
   02   subgroup-1.
      03 elem-2  PIC NNN.
      03   elem-3  PIC NN.
   02   sub-group-2.
      05   elem-4  PIC NNN.
      05   elem-5 PIC N(5).
```

Then, when a national group item is used, in general it is treated the same as an elementary national data item instead of being treated as an alphanumeric group item. Operations such as MOVE CORRESPONDING that operate on the individual elements of a group item will operate on the individual elements of a national group. Statements such as INSPECT, which could not be used correctly on alphanumeric groups containing national data, will operate correctly by treating the group as an elementary national data item.

When you move a data item of category alphanumeric to a data item of category national, or compare the two of them, the alphanumeric data item is automatically converted from usage DISPLAY to usage NATIONAL. The reverse is not true: to move a national item to an alphanumeric item, you need to use the DISPLAY-OF intrinsic function to convert the data. This results from an assumption that a national character set will typically include all the characters of an alphanumeric character set, but normally an alphanumeric character set will not include all the characters of a national character set.

D.11.3 Source code portability

Source code written in free-form reference format may be ported across systems utilizing character sets of differing character widths and different control function encodings with character set conversion, while source code written in fixed-form reference format might be more difficult to port because of its column-dependent specification.

D.12 Collating sequences

Collating sequences are used by the compiler during processing of a compilation-group and by the application at runtime. Traditionally, the same collating sequence was used in both cases and was typically defined by the bit pattern of characters in a coded character set. Now, it is becoming more widespread that the collating sequence used at runtime is not known at compile time. One reason for this is the development of applications that run in more than one country or culture. Users need collating sequences that are appropriate for their own language or culture. To meet this need, new features are provided in COBOL for selection of collating sequences at compile time and at runtime.

COBOL has two classifications of collating sequences:

— alphanumeric, and

— national.

An alphanumeric collating sequence applies to data described with usage DISPLAY. This name was chosen for its association with data of category alphanumeric, although some of the characters may be special characters or codes with no assigned graphic character. Alphanumeric data is typically, though not necessarily, represented in a 7-bit or 8-bit coded character set. To choose a collating sequence for data items described with usage DISPLAY, code the phrase COLLATING SEQUENCE FOR ALPHANUMERIC in various COBOL language constructs.

A national collating sequence applies to data described with usage NATIONAL. This name was chosen for its association with a 'national' language from the perspective of users in a given country (nation) or culture. This data is typically, though not necessarily, represented in a 16-bit coded character set. To choose a collating sequence for data items described with usage NATIONAL, code the phrase COLLATING SEQUENCE FOR NATIONAL in various COBOL language constructs.

These two classifications of collating sequences do not require two physically-separate collating sequence tables or implementations. A single collating sequence, such as the one associated with UCS-4, can be referenced as both an alphanumeric and a national collating sequence, and used to support both collating sequences. Similarly, a single locale can provide the collating sequence for the character repertoires of usage DISPLAY and usage NATIONAL.

D.12.1 Methods of defining collating sequences

Collating sequences for use in COBOL are defined in various ways:

1) the COBOL implementor may define collating sequences and give them a code-set name, and then a programmer can associate the code-set name with an alphabet-name in the SPECIAL-NAMES paragraph;

2) the COBOL programmer may define collating sequences in the SPECIAL-NAMES paragraph and give them an alphabet name;

3) collating sequences may be specified by a 'locale' outside COBOL and made available by the operating environment, and referenced by associating 'LOCALE' with an alphabet-name in the SPECIAL-NAMES paragraph;

4) a collating sequence may be defined by an international standard recognized by COBOL, and associated with an alphabet-name in the SPECIAL-NAMES paragraph; these are STANDARD-1 or STANDARD-2 for ISO/IEC 646, a 7-bit coded character set.

   ISO/IEC 646 does not define a collating sequence itself; COBOL specifies that the collating sequence is the order in which characters are defined in that coded character set.

5) collating sequences may be defined by the architecture of the processor used for compilation or execution of the application, and referenced by associating 'NATIVE' with an alphabet-name in the SPECIAL-NAMES paragraph.

Regardless of who defines a collating sequence and how it is defined, the common mechanism for identifying one is an alphabet-name. To use a collating sequence, the COBOL programmer gives it an alphabet-name in the SPECIAL-NAMES paragraph. The alphabet-name is then used in procedural code to reference the collating sequence.

D.12.2 Methods of selecting a collating sequence

D.12.2.1 Using the defaults

If you do nothing, the default alphanumeric and national collating sequences for comparisons and ordering are the NATIVE collating sequences, specified by the implementor and typically based on the bit pattern of characters in the coded character set used in the processor.

In some environments, that collating sequence may be culturally acceptable — the application might never need to be ported to another processor and the native collating sequences might be suitable for the users.

If the application is to be ported to other processors or is designed for users of differing cultures, the default collating sequences are unlikely to be suitable and you will need to design the application to use COBOL features that allow for selection of specific collating sequences or runtime determination of collating sequences.

D.12.2.2 Using a specific collating sequence

If you simply want to select a single alphanumeric collating sequence or a single national collating sequence, or one of each, for use throughout the entire compilation unit, code a PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph. This is illustrated in the following code fragment, using standard collating sequences:

```
PROGRAM-ID.   OrderParts.
    ...
ENVIRONMENT DIVISION.
    ...
OBJECT-COMPUTER.
PROGRAM COLLATING SEQUENCE FOR ALPHANUMERIC IS ASCII-Sort
                           FOR NATIONAL IS UCS-Sort.
    ...
SPECIAL-NAMES.
ALPHABET ASCII-Sort FOR ALPHANUMERIC IS STANDARD-1
ALPHABET UCS-Sort FOR NATIONAL IS UCS-4.
```

Nothing else is needed to use these collating sequences for comparisons, SORT, and MERGE throughout the OrderParts program.

D.12.2.3 Using a locale

The use of a locale for runtime selection of collating sequences is described in D.13.2.3, Locale-based collating sequences and in D.13.2.1.1, Switching locales in a COBOL runtime module.

D.12.2.4 Selecting a collating sequence for indexed files

The default collating sequence for primary and alternate record keys of indexed files is the native program collating sequence for the runtime module that created the file. A specific collating sequence can be selected by coding a COLLATING SEQUENCE clause in the file control entry for file creation and later access. Different collating sequences can be used for different keys; however, this capability is not widely supported by implementors.

The following code fragment illustrates a file control entry for specifying multiple alternate indexes, some having a unique collating sequence and others having the native collating sequence:

```
...
INPUT-OUTPUT SECTION.
FILE CONTROL.
    SELECT   File-1
        ACCESS MODE IS RANDOM
```



```
        ALTERNATE RECORD KEY IS   altkey_1
        ALTERNATE RECORD KEY IS   altkey_2
        ALTERNATE RECORD KEY IS   altkey_3
        ALTERNATE RECORD KEY IS   altkey_4
        RECORD KEY IS  prim-key
...
COLLATING SEQUENCE OF  altkey_1   altkey_2   FOR ALPHANUMERIC IS  Universal-order
COLLATING SEQUENCE OF  prim-key              FOR NATIONAL IS  Universal-order
...
```

This example is for a primary key described as usage NATIONAL and alternate keys described as usage DISPLAY. The collating sequence of the primary key and alternate record keys altkey_1 and altkey_2 is the one associated with alphabet-name Universal-order. Since there is no COLLATING SEQUENCE clause specifying altkey_3 or altkey_4, those keys are sequenced using the native collating sequence.

Selection of a collating sequence that equates two or more characters can be used to make keys match when the actual binary coding differs. For example, if lowercase letters were equivalenced to uppercase letters, a key containing "ABC" would be equal to "abc".

D.12.3 Compile-time collating sequences

The collating sequence known at compile-time is usually the native collating sequence of the processor for which the runtime module is being compiled. When THROUGH phrases are specified in a VALUE clause or in an EVALUATE statement, evaluation of the range of values might occur at compile time or at runtime, depending on the implementation. When the evaluation occurs at compile time, the actual range of values at runtime might not include the expected values because the runtime collating sequence might be different from the compile-time sequence. To deal with this, an alphabet-name can be specified in THROUGH phrases to ensure that the range of values at runtime is a specific set of values. For example:

```
ENVIRONMENT DIVISION.
...
SPECIAL-NAMES.
ALPHABET normal-range IS STANDARD-1.
...
DATA DIVISION.
...
1  a-condition PIC X.
88 ok VALUES ARE "1" THROUGH "G" IN normal-range.
```

results in a range of values specified by the collating sequence associated with STANDARD-1 (coded character set ISO/IEC 646) — in this case, 0-9, some special-characters, and uppercase A through G. The characters defined by this range of values will satisfy the condition 'ok' at runtime, regardless of the coded character set and collating sequence in use. This enhances portability of applications.

STANDARD-1 and UCS-4 are predefined alphabet-names that can be used to specify a useful portable range of values.

If you do not specify an alphabet-name, the range of values can be empty or can vary across processors because the default collating sequence chosen by the implementor is probably different across processors.

D.12.4 Intrinsic functions for comparisons

If there is no need for a culturally-appropriate collating sequence for all comparisons, but there is occasional need for it, use the comparison intrinsic function LOCALE-COMPARE, which compares two arguments using either the current locale or a specified locale.

If a sophisticated comparison suitable for multiple cultures is needed, use the comparison intrinsic function STANDARD-COMPARE, which compares two arguments using a standard ordering table that complies with ISO/IEC 14651. That ordering standard is not widely implemented at this writing, but support for it is expected to become more available over time.

D.13 Culturally-specific, culturally-adaptable, and multilingual applications

Culturally-specific applications are designed for the needs of one specific language or culture.

Culturally-adaptable applications are designed and coded once for users of many diverse languages and cultures, and are tailored at runtime to behave in ways suitable for a given language or culture.

Multilingual applications are designed and coded once to handle more than one language or culture in any given execution.

D.13.1 Culturally-specific applications

Features in this final committee draft International Standard that make it easier to develop applications for one specific language or culture are:

- multiple-character currency strings and mixed-case currency strings;
- a class test for characters in a particular alphabet;
- support for selecting and using a specific locale.

D.13.1.1 Currency string and currency symbol

The currency string is used at run time in input and output data to represent the monetary units associated with a numeric value.

The currency string is always specified as the object of the CURRENCY SIGN clause. It is used exactly as specified therein, subject only to any conversion from the computer's compile-time character set to the computer's run-time character set that might occur.

The currency symbol is used at compile time in PICTURE character-strings to indicate where and how to place the currency string in output data and where and how to expect it in input data.

When the PICTURE SYMBOL phrase is specified, the object of that phrase is the currency symbol. When the phrase is not specified, the object of the CURRENCY SIGN clause is the currency symbol (as well as the currency string).

When comparing currency symbols, uppercase and lowercase basic letters are equivalent, uppercase and lowercase extended letters may be equivalent, and basic and extended letters may be equivalent.

Consider the following program elements:

Example 1: CURRENCY SIGN 'm'. ...

PICTURE mMm,MmM,mM9.99-.

Example 2: CURRENCY SIGN '$' PICTURE SYMBOL 'q'. ...

PICTURE QqQ,qQq,Qq9.99'.

In Example 1, 'm' is both the currency string and the currency symbol. The value -123,456.78 would appear as ' m123,456.78-' in data associated with that picture clause.

In Example 2, '$' is the currency string and 'q' is the currency symbol. The value -876,543.21 would appear as ' $876,543.21-' in data associated with that picture clause.

D.13.1.2 Class test for characters in a particular alphabet.

To define a multiple-character currency string, the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph needs to be coded; for example:

```
SPECIAL-NAMES.
CURRENCY-SIGN IS "EUR " WITH PICTURE SYMBOL "u".  *> note the space after EUR
```

Then the letter "U" can be used in picture character strings; for example: PIC U99.99 (which defines an item of length 9). For the value 10.00, the edited data item would contain 'EUR 10.00'.

The following code fragment illustrates a class test using the coded character set associated with the name CYRILLIC, assuming the implementor has defined the name CYRILLIC and the associated coded character set:

```
SPECIAL-NAMES.
ALPHABET ok-data FOR ALPHANUMERIC IS CYRILLIC.
...
DATA DIVISION.
01   the-input-stream  USAGE DISPLAY   PIC  N(80).
...
PROCEDURE DIVISION.
...
IF  the-input-stream  IS  ok-data  THEN  ...
...
```

This class test is true if all characters in the-input-stream are characters specified in the coded character set CYRILLIC.

D.13.2 Culturally-adaptable applications

The nature of a culturally-adaptable application is that it is written once and there is no need to recompile the source code in order for the application to be used on the same type of processor in different cultures. COBOL supports cultural adaptability for the following cultural elements:

- monetary formatting
- number formatting
- collating sequences of file indexes, sort/merge, and comparisons
- case classification of letters
- date and time formatting

To make an application culturally-adaptable, the details of these cultural elements are specified outside of COBOL where they can be easily selected at runtime. The details are specified in a "locale", which contains a set of cultural conventions typically constructed by a utility in the operating environment, supplied by the COBOL implementor, or supplied with the operating environment. The implementor may provide such a utility, it may come with the operating environment, or it may have to be purchased. The implementor may also provide some pre-built locales. Then, in each operating environment that supports locales, there is a system locale defined for the local culture and, typically, the capability for each application to have a user locale and even a variety of locales that can be selected during execution of the application.

This final committee draft International Standard requires that at least one locale be provided by COBOL implementors in cases where the operating environment does not support locales. The user will need to know which locales are available to the application.

In designing an application for cultural adaptability, there is much more to consider than just the cultural elements supported by COBOL. For example, it may be necessary to choose field sizes that accommodate a variety of monetary values. It will help if message text is isolated in tables or files that can be easily substituted. It is necessary to think about the elements that can vary across the cultures for which you are designing, and about how the details of those elements can be determined at runtime rather than at compile time.

D.13.2.1 Locale selection

At startup, a user-default locale is available to the run unit and will be used for locale-based processing for all locale categories in every COBOL runtime module in that run unit -- unless a specific locale is indicated for a given data element, function, or statement — until a SET statement is executed to switch to another locale.

A SET statement can be used to switch locales for a single locale category or for all categories at once. For example, if the SYSTEM-DEFAULT locale is to be used just for numeric formatting, a SET statement can be coded as follows:

```
SET LOCALE LC_MONETARY TO SYSTEM-DEFAULT
```

Other examples of locale selection are given below.

D.13.2.1.1 Switching locales in a COBOL runtime module

If a specific locale or multiple specific locales are to be used in an application, instead of the system default or user default locales, the locale needs to be identified in the SPECIAL-NAMES paragraph and a locale-name assigned for use in COBOL procedural code. The details of how to identify a locale are specified by the COBOL implementor. A library of named locales might be provided, or an implementor could accept a data set location in a literal -- these are examples; see the user documentation for your COBOL compiler to find out what your implementor has provided.

The following code fragment illustrates the code for identifying and switching locales, assuming the implementor has provided locales named French-1 and Swiss-2:

```
PROGRAM-ID.   IndexParts.
ENVIRONMENT DIVISION.
OBJECT-COMPUTER.
PROGRAM COLLATING SEQUENCE FOR NATIONAL IS Locale-Sort.
*>  This causes the national collating sequence to be determined at runtime
*>  by the current locale -- the user-default unless a SET LOCALE statement has
*>  switched to another locale.
       ...
SPECIAL-NAMES.
*>  Name some locales for use later in SET LOCALE statements:
    LOCALE  French IS "D:\LocaleLib\French-1"  *>  library example
    LOCALE  Swiss-2  IS  Swiss                 *>  another example, a named locale
       ...
*>  Associate the alphabet-name "Locale-Sort" with the locale facility:
    ALPHABET Locale-Sort FOR NATIONAL IS LOCALE.
...
DATA DIVISION.
...
01  default-locale-pointer  USAGE POINTER.  *> for saving the default locale
...
PROCEDURE DIVISION.
*>  The user-default locale is in effect initially.  If you will need it again, save a
*>  pointer identifying the user-default locale:
    SET LOCALE default-locale-pointer TO LOCALE USER-DEFAULT.
        ...
*>  Now, switch to the locale you want to use.
*>  If you don't want to switch locales for all locale
*>  categories, specify a SET statement for each category you want to switch:
    SET LOCALE  LC_COLLATE  TO  Swiss.    *>  switch program collating sequence
    SET LOCALE  LC_MONETARY  TO  French   *>  switch monetary formatting
...
*>  Suppose now you have completed processing with the two locales and you want to use
*>  the user default for a while:
    SET LOCALE  LC_COLLATE  TO  USER-DEFAULT
    SET LOCALE  LC_MONETARY  TO  USER-DEFAULT
        ...
*>  Now, suppose you want to switch the user default to one of your locales:

    SET LOCALE  USER-DEFAULT  TO  Swiss
       ...
*>  Now, having finished your work with the Swiss locale, restore the previous
*>  USER-DEFAULT locale:

    SET LOCALE USER-DEFAULT TO default-locale-pointer.
        ...
```

It is always a good idea to save the address of the current locale before switching to a new one, unless it is known that no further processing will need that locale.

D.13.2.1.2 Switching locales outside of COBOL

The implementor may, but is not required to, support inter-operation of COBOL with other programming languages; one example is the C programming language. When support is provided, the implementor decides whether a locale switch for the user-default locale in an activated non-COBOL runtime module is capable of being recognized on return to COBOL. If supported, recognition in COBOL is not automatic; execution of a SET statement specifying the user-default locale is required in COBOL; for example, to access all categories of the new locale:

```
SET LOCALE LC_ALL TO USER-DEFAULT.
```

D.13.2.2 Locale-based monetary and numeric formatting

Locale-based monetary or numeric formatting is done by specifying the LOCALE phrase in the PICTURE clause in the data description entry of a data item.

To have an item edited entirely in accordance with the locale specification, regardless of its picture specification, a LOCALE phrase must be coded in the picture character string. When using the LOCALE phrase, the picture character string is not an indication of the field size needed to hold the edited item. The programmer should design for the largest size needed and code a SIZE phrase in the PICTURE clause.

For example:

```
01 US-amount PICTURE +$9.9 LOCALE SIZE IS 10.
```

results in a signed numeric-edited field 10 characters long that will be edited in accordance with locale category LC_MONETARY. The currency string and its placement, the sign convention, the grouping separator and placement, and the decimal separator and placement will be determined from the locale that is current at the time of editing. The locale category LC_MONETARY is used for all numeric formatting, both for monetary and non-monetary formatting.

If a specific locale is to be used for editing, a locale-name needs to be assigned to it in the SPECIAL-NAMES paragraph, for example, my-locale in the following:

```
SPECIAL-NAMES.
    LOCALE my-locale is USA-1.
*>  assuming the implementor has given you a way to define or reference a locale
*>  named USA-1
...
DATA DIVISION.
...
01  US-amount PICTURE +$9.9 LOCALE my-locale SIZE IS 10.
```

Multiple locales may be used; for example, if a report had US currency in one column and Italian currency in another, one could define:

```
LOCALE my-US-locale is USA-1.
LOCALE my-Italian-locale is Italian-1. *> assuming locales named USA-1 and Italian-1
...
DATA DIVISION.
01  REPORT-DETAIL.
...
05  US-amount PICTURE +$9.9 LOCALE my-US-locale SIZE IS 8.
...
05  Italian-amount PICTURE +$9.9 LOCALE my-Italian-locale SIZE IS 18.
...
```

D.13.2.3 Locale-based collating sequences

To use a collating sequence specified by a locale throughout the entire compilation unit, a PROGRAM COLLATING SEQUENCE clause must be coded specifying an alphabet-name that is associated with the locale facility. The following code fragment illustrates the selection of a locale for a national collating sequence and the selection of a standard alphanumeric collating sequence:

```
OBJECT-COMPUTER.
    PROGRAM COLLATING SEQUENCE FOR ALPHANUMERIC IS ASCII-Sort
                               FOR NATIONAL IS UCS-Sort.
      ...
SPECIAL-NAMES.
    ALPHABET ASCII-Sort FOR ALPHANUMERIC IS STANDARD-1
    ALPHABET UCS-Sort FOR NATIONAL IS LOCALE
```

This causes the runtime national collating sequence to be the collating sequence specified by locale category LC_COLLATE in the locale current during execution. The alphanumeric collating sequence is the collating sequence associated with STANDARD-1.

The following code fragment illustrates use of a single locale-based collating sequence in the SORT statement:

```
PROGRAM-ID. IndexParts.
...
SPECIAL-NAMES.
    ALPHABET ucs-sort-a FOR ALPHANUMERIC IS LOCALE
    ALPHABET ucs-sort-n FOR NATIONAL IS LOCALE
. . .
PROCEDURE DIVISION.
...
*> Sort a file using the current locale for both alphanumeric and
*> national keys
    SORT SortFile ASCENDING KEY key-item-a, key-item-n
      COLLATING SEQUENCE
        FOR NATIONAL IS ucs-sort-n
        FOR ALPHANUMERIC IS ucs-sort-a
      USING A-file
      GIVING Another-file.
...
```

The switching of locales used for collating sequences is illustrated in D.13.2.1.1, Switching locales in a COBOL runtime module.

D.13.2.4 Locale-based case classification of letters

It is possible to have the case classification of letters determined by locale category LC_CTYPE, which defines character classification, case conversion, and other character attributes. To do this, a CHARACTER CLASSIFICATION clause must be specified in the OBJECT-COMPUTER paragraph. For example:

```
OBJECT-COMPUTER.
    CHARACTER CLASSIFICATION IS GERMAN, JAPANESE.
    *> GERMAN applies to USAGE DISPLAY, JAPANESE to USAGE NATIONAL
...
SPECIAL-NAMES.
    LOCALE GERMAN IS "GER046"
    LOCALE JAPANESE IS Japan-123.
```

This results in runtime use of the specified locales for determining

— whether characters are ALPHABETIC, ALPHABETIC-UPPER, or ALPHABETIC-LOWER in a class test;

— whether the content of a data item is consistent with its picture character-string when checked with a VALIDATE statement;

— the case of characters for conversion using the UPPER-CASE intrinsic function or the LOWER-CASE intrinsic function.

Locale-based case conversion with the UPPER-CASE function or the LOWER-CASE function can result in the return of more or fewer characters than the argument, although the use of these functions without a locale always returns a string that is the same length as the argument.

D.13.2.5 Date and time formatting

Locale-based date and time formatting are provided by the intrinsic functions LOCALE-DATE and LOCALE-TIME, respectively.

To convert a given date field to a locale-based format in accordance with a specific locale, use the LOCALE-DATE function and specify the locale's locale-name as the second argument; for example:

```
MOVE FUNCTION LOCALE-DATE (some-date-field, my-italian-locale) TO  a-date-field
```

To obtain the current date in a locale-based format in accordance with the current locale, use the LOCALE-DATE function without specifying a locale as the second argument; for example:

```
MOVE FUNCTION LOCALE-DATE (CURRENT-DATE (1:8)) TO a-date-field.
```

Similarly, you can obtain locale-based time using the LOCALE-TIME function.

D.13.3 Multilingual applications

Multilingual applications are more easily developed if an implementor supports one of the formats of the UCS as a runtime coded character set. The UCS supports most languages of the world, and characters for more languages are being added as development of the UCS continues.

At this writing, devices that support UCS-4 and UTF-16 are not yet widely available, but their availability is expected to increase over time because of the need for applications of international scope. The user needs to check the implementor's documentation to determine whether the UCS is used as a runtime coded character set, either for usage DISPLAY or for usage NATIONAL. Typically, it will be used for usage NATIONAL.

Support for Unicode® is synonymous with support for the UCS.

Use of UCS-4 or UTF-16 alone is not always sufficient for developing multilingual applications. If the application requires culturally-correct ordering, monetary or number formatting, date and time formatting, or case classification of letters, use of the locale-based features described in D.13.2, Culturally-adaptable applications, may be necessary.

D.14 External switches

An external switch is a hardware or software device, defined and named by the implementor, that is used to indicate that one of two alternate states exists. These alternate states are referred to as the on status and the off status of the associated external switch.

The status of an external switch may be interrogated by testing condition-names associated with that switch. The association of a condition-name with an external switch and the association of a user-specified mnemonic-name with the switch-name of an external switch is established in the SPECIAL-NAMES paragraph of the environment division.

The status of certain switches may be altered by the SET statement.

D.15 Common exception processing

Exception processing is a method for detecting and processing exceptions that occur during the execution of COBOL statements.

There is more than one method of exception processing. The classical methods are: specifying exception phrases on various statements, such as AT END, INVALID KEY, ON EXCEPTION, and so on; checking status values using FILE STATUS; and invoking USE statements based on i-o status codes, open modes, and file-names. All of the classical methods are always enabled, take precedence over common exception processing, and work in exactly the same way they have in previous COBOL standards. The other method of exception processing is called

common exception processing. It is based on exception conditions and can be enabled and disabled, depending on a compiler directive. Common exception processing is described in the following paragraphs.

An exception may be due to an error or due to some condition arising during the processing of a statement. When an exception occurs, the associated exception condition exists. When an exception condition exists, further processing takes place as described in the following paragraphs.

Associated with each exception condition is an exception-name or an exception object. The concepts of exception objects are given in D.18.9, Exception objects. The following refers to the predefined exception conditions represented by exception-names. Syntax for processing exception conditions uses these exception-names. They may be specified only in the TURN compiler directive, the RAISING phrase of the EXIT or GOBACK statement, the RAISE statement, and the USE statement. There are three levels of exception-names. Level-1 is one all-inclusive exception name, EC-ALL. Level-2 identifies exceptions associated with a specific type of exception. The level-2 exception-names are: EC-ARGUMENT, EC-BOUND, EC-DATA, EC-FLOW, EC-I-O, EC-IMP, EC-LOCALE, EC-OO, EC-ORDER, EC-OVERFLOW, EC-PROGRAM, EC-RAISING, EC-RANGE, EC-REPORT, EC-SCREEN, EC-SIZE, EC-SORT-MERGE, EC-STORAGE, EC-USER, and EC-VALIDATE. The level-3 exception-names are the level-2 exception-names suffixed by a descriptive character-string that identifies that actual exception condition. When a level-3 exception condition is raised, the associated level-2 and EC-ALL can be used to process the exception, if desired.

The user can define exceptions by suffixing EC-USER-. For example, EC-USER-OVERDRAWN might mean that an account is overdrawn. By executing 'RAISE EC-USER-OVERDRAWN' the user could cause a declarative to be executed.

In a similar fashion, the implementor can suffix EC-IMP- to define exceptions. In this case the implementor defines the fatality, what causes the exception and so on.

Checking for exceptions is initially disabled for all exception conditions, and can be enabled at compile time with the use of the TURN compiler directive. If checking for an exception condition is enabled, it can also be disabled by the TURN compiler directive. The TURN compiler directive may specify the level-3 exception-name, the associated level-2 exception-name, or EC-ALL.

Checking for an exception condition is locally disabled by the presence of an explicit phrase. For example, if the SIZE ERROR phrase is specified on an arithmetic statement, the raising of the EC-SIZE exception condition is disabled for that statement, except during item identification.

When an exception condition exists and checking for that exception condition is enabled, the exception condition is raised and the last exception status is set to indicate that exception condition. Subsequent processing depends on the presence of explicit phrases or an applicable exception declarative. If there is an explicit phrase, that phrase takes precedence over any declarative or default action. For example, when an arithmetic size error occurs, if an explicit SIZE ERROR phrase is specified, that phrase takes precedence and the EC-SIZE exception condition is not raised. If there is no such phrase, any declarative specified with USE AFTER EXCEPTION EC-SIZE would be executed. The SIZE ERROR and NOT SIZE ERROR phrases will function even if checking for EC-SIZE is not enabled.

If an exception declarative is executed, there are several ways to terminate execution of that declarative. The user can execute 'EXIT ... RAISING' or 'GOBACK RAISING' to cause the exception that caused the declarative to be executed (RAISING LAST EXCEPTION) or another exception ('RAISING EXCEPTION exception-name' or 'RAISING identifier') to be propagated to the activating function, method, or program. This causes that exception condition to exist in the activating runtime element. In addition, the user can execute a RESUME statement to cause execution to be continued at the statement following the statement that caused the exception to be executed (RESUME AT NEXT STATEMENT) or a RESUME statement to cause execution to be continued at a procedure-name that is in the nondeclarative portion of the source element (RESUME AT procedure-name). The final method is to fall through the last procedure in the declarative. When this is done, if the exception condition is a fatal exception condition the execution of the run unit is terminated. If the exception condition is not fatal, execution continues as if RESUME AT NEXT STATEMENT were executed.

The default action taken when an exception condition is enabled, the exception exists, no applicable declarative exists, and no explicit phrase is specified on the statement depends on the specific exception condition and other

factors. If the exception condition is defined to be nonfatal, execution continues as specified in the rules for the statement. For example, EC-I-O-AT-END will cause execution to continue at the next executable statement following the READ statement. If the exception condition is defined to be fatal, further processing depends on the PROPAGATE compiler directive. If PROPAGATE ON is in effect, execution of the current runtime element is terminated and the exception that occurred is propagated to the activating runtime element, as if an EXIT statement or GOBACK statement with the RAISING LAST EXCEPTION phrase were specified. If PROPAGATE ON is not in effect, execution of the run unit is terminated.

The user can cause an exception to be raised by executing the RAISE statement, primarily for user-defined exceptions.

Additional information about an exception condition is obtained through the use of a series of functions. These functions return detailed information about the last exception status. The functions and their returned values are:

EXCEPTION-FILE returns an alphanumeric character string that contains information about the last I-O status value and any file connector that was associated with the last exception status.

EXCEPTION-FILE-N returns a national character string that contains information about the last I-O status value and any file connector that was associated with the last exception status.

EXCEPTION-LOCATION returns an alphanumeric character string that indicates the location of the statement in which the exception condition associated with the last exception status was raised. Part of the string is implementor-defined.

EXCEPTION-LOCATION-N returns a national character string that indicates the location of the statement in which the exception condition associated with the last exception status was raised. Part of the string is implementor-defined.

EXCEPTION-STATEMENT returns the name of the statement in which the exception condition associated with the last exception status was raised.

EXCEPTION-STATUS returns the exception-name associated with the last exception status.

D.16 Rounding

COBOL provides the capability of specifying rounding in arithmetic statements and expressions at various points in the evaluation process and as values are prepared for storing into receiving data items.

The following forms of rounding are provided (examples presume an integer destination):

— AWAY-FROM-ZERO: Rounding is to the nearest value of larger magnitude.

— NEAREST-AWAY-FROM-ZERO: Rounding is to the nearest value. If two values are equally near, the value with the larger magnitude is selected. This mode has historically been associated with the ROUNDED clause in previous versions of standard COBOL.

— NEAREST-EVEN: Rounding is to the nearest value. If two values are equally near, the value whose rightmost digit is even is selected. This mode is sometimes called "Banker's rounding".

— NEAREST-TOWARD-ZERO: Rounding is to the nearest value. If two values are equally near, the value with the smaller magnitude is selected.

— PROHIBITED: No rounding is performed. If the value cannot be represented exactly in the desired format, the EC-SIZE-TRUNCATION condition is set to exist and the results of the operation are undefined.

— TOWARD-GREATER: Rounding is toward the nearest value whose algebraic value is larger.

— TOWARD-LESSER: Rounding is toward the nearest value whose algebraic value is smaller.

— TRUNCATION: Rounding is to the nearest value whose magnitude is smaller. This mode has historically been associated with the absence of the ROUNDED clause as well as for the formation of intermediate results in the prior COBOL standard.

Table C.3, ROUNDED MODE examples, illustrates the effect of the various rounding modes with various values. The ellipses in the column heading values indicate repetitions of the last digit in the intermediate result. The column contents indicate the result expected when rounding that intermediate result value to an integer.

Table C.3 — ROUNDED MODE examples

	+2.49	**-2.49**	**+2.50**	**-2.50**	**+3.49**	**-3.49**	**+3.50**	**-3.50**	**+3.510**	**-3.510**
Away-from-zero	+3	-3	+3	-3	+4	-4	+4	-4	+4	-4
Nearest-away-from-zero	+2	-2	+3	-3	+3	-3	+4	-4	+4	-4
Nearest-even	+2	-2	+2	-2	+3	-3	+4	-4	+4	-4
Nearest-Toward-Zero	+2	-2	+2	-2	+3	-3	+3	-3	+4	-4
Toward-Greater	+3	-2	+3	-2	+4	-3	+4	-3	+4	-3
Toward-Lesser	+2	-3	+2	-3	+3	-4	+3	-4	+3	-4
Truncation	+2	-2	+2	-2	+3	-3	+3	-3	+3	-3

The programmer may specify how individual intermediate values are rounded when they are stored into receiving data items through the ROUNDED clause; may select a default mode of rounding to be used when the ROUNDED clause appears with no further qualification on a receiving data item through the DEFAULT ROUNDED MODE clause of the OPTIONS paragraph of the IDENTIFICATION DIVISION; and may specify how arithmetic operations and conversions to and from intermediate forms are rounded through the INTERMEDIATE ROUNDING clause.

D.16.1 Intermediate rounding

Intermediate rounding applies when data items are retrieved for inclusion in an arithmetic operation and during the execution of arithmetic operations to produce an intermediate result.

In the previous edition of the COBOL standard, for multiplication and division in standard arithmetic, the default mode of rounding for inexact results was truncation to 32 significant digits. This default is unchanged in this final committee draft International Standard, and is also the default for standard-binary and standard-decimal arithmetic.

When the intermediate value can be represented exactly in the appropriate intermediate format, the exact value is used.

In the event the value cannot be exactly represented, the user may now specify other modes of rounding for arithmetic operations and for the conversions to and from intermediate forms used in the arithmetic operations through the optional INTERMEDIATE ROUNDING clause of the OPTIONS paragraph of the IDENTIFICATION DIVISION.

Specifically, the following options are available in the INTERMEDIATE ROUNDING clause:

— NEAREST-AWAY-FROM-ZERO
— NEAREST-EVEN
— PROHIBITED
— TRUNCATION

for which the descriptions are found in D.16, Rounding.

If the INTERMEDIATE ROUNDING clause is not specified, INTERMEDIATE ROUNDING IS TRUNCATION is implied. The implicit intermediate rounding specified in this final committee draft International Standard is unchanged from previous editions of Standard COBOL.

D.16.2 Final rounding (the ROUNDED clause)

Final rounding applies to the formation of the final result of the expression or statement immediately before the result is placed in the destination. This form of rounding is that which is associated with the ROUNDED clause.

In previous COBOL standards, only two methods of "final" rounding were provided:

— Rounding toward the smaller magnitude (truncation, signaled by the absence of the ROUNDED clause)

— Rounding to the nearest value; if two values were equally near, the value with the larger magnitude was chosen (signaled by the presence of the ROUNDED clause).

The ROUNDED clause has been enhanced to provide explicit selection of rounding modes, for example, ROUNDED MODE IS NEAREST-EVEN. The rounding modes available in the ROUNDED clause are described in D.16, Rounding.

If the ROUNDED clause is not present for a given result, the rules for ROUNDED MODE IS TRUNCATION apply. This provides the same behavior as specified in the previous COBOL standard.

The optional DEFAULT ROUNDED MODE clause in the OPTIONS paragraph of the IDENTIFICATION DIVISION is provided to allow the user to specify the mode of rounding for any operation for which the ROUNDED phrase is specified without the MODE phrase. The default rounding modes that may be specified are described in D.16, Rounding.

If the DEFAULT ROUNDED MODE clause is not specified in the program, the effect of the ROUNDED phrase without the MODE phrase is as if ROUNDED MODE IS NEAREST-AWAY-FROM-ZERO had been specified. This provides the same functionality for rounding as was provided in prior COBOL standards, and is also the same functionality provided by DEFAULT ROUNDED MODE IS NEAREST-AWAY-FROM-ZERO.

If the DEFAULT ROUNDED MODE clause appears, ROUNDED phrases without MODE phrases are treated as if they had been specified with the rounding mode specified in the DEFAULT ROUNDED MODE clause.

D.17 Forms of arithmetic

COBOL provides four basic forms of arithmetic: native, standard, standard-binary and standard-decimal.

Native arithmetic allows the implementor to use intermediate data item formats and computational techniques that the implementor deems most efficient for a particular operating environment.

Standard arithmetic is an obsolete feature of standard COBOL.

Standard-binary arithmetic provides most of the advantages of standard arithmetic, slightly greater inherent precision, a specific format for arithmetic operands and intermediate results, and arithmetic rules and formats explicitly described in a platform-independent, language-independent standard. In standard-binary arithmetic, the arithmetic operand format is a binary floating-point format, and for decimal operands there is some risk of precision loss during the conversion from binary floating-point form to decimal and vice versa.

Standard-decimal arithmetic provides all of the advantages of standard-binary arithmetic, and in addition provides a platform-independent, language-independent decimal floating-point format, thereby avoiding the conversion precision loss that might be entailed for decimal operands in standard-binary arithmetic.

For brevity, throughout D.17, Forms of arithmetic, "IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic" shall be referred to simply as "IEEE 754-2008".

The ARITHMETIC clause in the identification division determines the mode of arithmetic for the source unit: native arithmetic, standard arithmetic, standard-decimal arithmetic or standard-binary arithmetic. If the ARITHMETIC clause is not specified, native arithmetic is the mode of arithmetic for the source unit.

Most numeric operands in COBOL have historically been in some decimal format.

When standard-decimal arithmetic is in effect, intermediate results are in a decimal form and must be converted from IEEE 754-2008 Decimal128 to IEEE 754-2008 Binary128 format before the values can be stored in binary destinations.

Similarly, when standard-binary arithmetic is in effect, intermediate results must be converted from IEEE 754-2008 Binary128 format to IEEE 754-2008 Decimal128 format before the values can be stored in decimal destinations.

Radix conversions can cause precision loss and may be costly in performance. Results using decimal sending and receiving operands are likely to be more precise using standard-decimal arithmetic than standard-binary arithmetic precisely because radix conversions are avoided.

The following characteristics are common to standard arithmetic, standard-binary arithmetic, and standard-decimal arithmetic:

1) A single format appropriate to the specified arithmetic mode is used for each operand in an arithmetic expression and the result of every arithmetic operation, arithmetic expression, and integer and numeric intrinsic functions.

2) The binary arithmetic operators +, -, * and / and the SQRT function are defined to give results that are accurate to the precision of the format appropriate to the arithmetic mode. Exponentiation is defined to give results that are accurate to the precision of that format for exponents with the values –4, -3, -2, -1, 0, 1, 2, 3, and 4.

3) The size error condition exists when the result of any single arithmetic operation cannot be contained in the format appropriate to the arithmetic mode.

The size error condition exists when certain division and exponentiation errors occur and when the resultant identifier of an arithmetic statement cannot contain the value in the intermediate result.

D.17.1 Standard arithmetic

Standard arithmetic was introduced in the previous edition of the standard. It is both an obsolete element of standard COBOL and a processor-dependent feature of the language. Technological improvements in the industry, in particular the rules detailed in IEEE 754r regarding rounding and arithmetic operations, and the 128-bit decimal and binary floating-point formats in that standard, have outstripped the capabilities and the potential portability of standard arithmetic.

These superior mechanisms are represented by standard-decimal and standard-binary arithmetic, discussed in D.17.2, Standard-decimal arithmetic, and D.17.3, Standard-binary arithmetic, respectively.

When standard arithmetic is in effect most common arithmetic operations will produce results that are predictable, reasonable, and portable. In this context, portable means that the results will be identical from implementation to implementation.

The following major decisions are incorporated in standard arithmetic. Specific rationale is outside the scope of this document, but the general considerations include importance of functionality to the using community, cost of execution, and results that are reasonable and expected.

1) The standard intermediate data item contains 32 significant digits.

2) Zero is a unique value of a standard intermediate data item.

3) A standard intermediate data item is described in the form of a floating-point decimal number with a normalized fraction, although the representation may be in any form that provides the same results.

4) The exponent of a standard intermediate data item can range from 999 through -999, inclusive.

5) A result that is close to zero but is not representable in a standard intermediate data item causes a size error condition to be raised and an EC-SIZE-UNDERFLOW exception condition to be set to exist instead of being rounded to zero.

6) In each binary operation and the SQRT function, digits beyond the thirty-second significant digit are truncated. Truncation is a form of rounding that is also referred to as "rounding to zero."

7) Before a compare involving one or more standard intermediate data items, the result is rounded to 31 digits.

8) In exponentiation, even if the base is portable only exponent values of –4, –3, ..., 3, 4 produce portable results.

When standard arithmetic is specified, regardless of whether the results are portable, the results exhibit the following two types of predictability.

1) Within a single execution of a runtime element, arithmetic statements, arithmetic expressions, the SUM clause, and numeric and integer intrinsic functions will yield the same arithmetic results, so long as the value and order of the operands or arguments are the same.

2) Equivalent methods of expressing certain arithmetic produce the same arithmetic result. This is true when the equivalent methods are defined in terms of arithmetic expressions. For example, the following three statements all yield the same arithmetic result:

```
ADD a b c GIVING d
COMPUTE d = FUNCTION SUM (a b c)
COMPUTE d = a + b + c
```

   Under standard arithmetic the first two are explicitly defined as being equivalent to the arithmetic expression (a + b + c) which is the same expression given in the third example.

It is important to note that the order of operands and operators can be significant. This is especially true when there is a great difference in the magnitude of the operands in an expression. For example the following two statements may yield different results:

```
ADD a b c GIVING d
ADD c b a GIVING d
```

Another point to be noted is that when standard arithmetic is specified, if any part of an arithmetic expression is implementor-defined then the remainder of the arithmetic expression is still evaluated according to the applicable rules. For example, in the expression

```
13 + A ** 0.4
```

whatever the implementor-defined value of A ** 0.4 is, the result of the entire expression will be exactly the result of adding 13 to that value according to the rules for evaluating arithmetic expressions. If the value of A ** 0.4 is 103.698974 then the value of the full expression will be 116.698974.

D.17.1.1 Specification

The ARITHMETIC IS STANDARD clause in the options paragraph of the identification division specifies that the mode of arithmetic for the source unit is standard arithmetic.

D.17.2 Standard-decimal arithmetic

Standard-decimal arithmetic is preferred over standard arithmetic (described at D.17.1, Standard arithmetic), as well as over standard-binary arithmetic (described at D.17.3, Standard-binary arithmetic).

Standard-decimal arithmetic shares with standard arithmetic the specification that arithmetic operations are performed, and intermediate results produced, in a decimal floating-point format, which is an advantage because most numeric operands in COBOL have historically been in some decimal format, and the performing of arithmetic operations using a floating-point decimal format eliminates conversion from and to binary forms, with the possibility of precision loss in that process, when the operands themselves are decimal.

The format for a standard-decimal intermediate data item is explicitly defined in IEEE 754-2008 (as the Decimal128 format). This format is identical to that of a data item declared USAGE IS FLOAT-DECIMAL-34, which means the COBOL standard provides for the ability of a user to declare a data item in the same format.

This contrasts with standard arithmetic in that there is no USAGE in standard COBOL that directly corresponds to a "standard intermediate data item", and in that the implementor defines the actual format of such a data item.

The rules for addition, subtraction, multiplication and division and the SQRT intrinsic function are explicitly defined in IEEE 754-2008, and COBOL uses the definitions in that industry standard rather than relying on language-specific rules for these operations.

Likewise, the specifications as to how intermediate and final rounding is performed either explicitly cite the corresponding rules in IEEE 754-2008 or use them as their basis.

The major motivation for this enhancement was to improve the potential portability of COBOL applications by having COBOL produce results consistent with existing, platform-independent, language-independent standards, specifically, IEEE 754-2008. Results produced in conformance with that standard are well-defined. It is anticipated that vendors will implement efficient means of storing and operating upon data items in IEEE 754r formats, leading to the expectation that such implementations will also be efficient.

The use of IEEE 754-2008 specifications in support of COBOL data types, arithmetic rules, and rounding and truncation algorithms, helps the user to control and minimize imprecision.

The primary advantage of standard-decimal arithmetic over standard-binary arithmetic in environments in which both are available is that no radix conversion is needed for decimal sending and receiving operands, and thus no associated precision loss occurs.

D.17.2.1 Specification

The ARITHMETIC IS STANDARD-DECIMAL clause in the options paragraph of the identification division specifies that the mode of arithmetic for the source unit is standard arithmetic.

D.17.2.2 Examples

These examples demonstrate a few of the features and requirements of standard-decimal arithmetic.

In the following examples, "sbidi" stands for standard-binary intermediate data item, "sdidi" stands for standard-decimal intermediate data item, "ulp" is an abbreviation for "unit in last position", and "ir-n" stands for the "nth intermediate result".

"Rounding" in the examples below implies the application of the MODE phrase of the INTERMEDIATE ROUNDING clause according to the rules of IEEE 754-2008. The application of the ROUNDED clause is explicitly described in the examples.

```
1  A  pic s9(4) value +8000.
1  B  pic s9(4) value +3000.
1  C  pic s9(4) value –4001.
1  D  pic s9(4) value is zero.
1  E  pic s99V99 value is +56.79.
1  F  usage float-binary-34.
1  G  usage float-decimal-34.
```

```
COMPUTE D = A + B + C
    *> begin
    *> convert A to sdidi  as ir-1
    *> convert B to sdidi  as ir-2
    *> add ir-1, ir-2 with rounding, result in ir-1
    *> convert C to sdidi as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> move ir-1 to D (truncating fractional digits)
    *> end


COMPUTE D ROUNDED = D + E
    *> begin
    *> convert D to sdidi  as ir-1
    *> convert E to sdidi  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> align decimal point of ir-1 to that of D
    *> adjust ulp according to DEFAULT ROUNDED MODE clause
    *> move adjusted ir-1 to D
    *> end


COMPUTE F = A + B + C
    *> begin
    *> convert A to sdidi  as ir-1
    *> convert B to sdidi  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> convert C to sdidi  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> convert ir-1 to sbidi format with rounding  as ir-2
    *> move ir-2 to F (ir-2 and F are in the same format)
    *> end


COMPUTE G ROUNDED = G + E
    *> begin
    *> retrieve G (it is already in sdidi format) as ir-1
    *> convert E to sdidi  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> align decimal point of ir-1 to that of G
    *> adjust ulp according to DEFAULT ROUNDED MODE clause
    *> move adjusted ir-1 to G
    *> end
```

D.17.3 Standard-binary arithmetic

Standard-binary arithmetic is provided as an alternative and as an adjunct to standard-decimal arithmetic, described at D.17.2, Standard-decimal arithmetic.

Standard-binary arithmetic operations will produce results that are predictable, reasonable, and portable. In addition, the format of the intermediate results and the rounding rules associated with intermediate operations are defined in an external standard. This feature provides for arithmetic results that are not only identical from implementation to implementation but also are defined by an international standard intended for platform-independent implementation. Standard-binary arithmetic shares these advantages with standard-decimal arithmetic.

However, the radix conversions required to convert decimal sending operands into standard-binary intermediate operand format, as well as those required to convert the intermediate values into standard-decimal intermediate data items intermediate values must be converted from binary to decimal form before they can be stored in decimal operands.

Radix conversions are by nature often imprecise; they also introduce complexity into the algorithm that is absent when the calculations as well as the operands are decimal. In an environment like COBOL in which operands are so often in a decimal form, the results will be more accurate using standard-decimal arithmetic than with standard-binary, precisely because those radix conversions are unnecessary for those operands in the former case.

It is also likely that these additional radix conversions will require execution-time resources that would not otherwise be needed.

D.17.3.1 Specification

The ARITHMETIC IS STANDARD-BINARY clause in the options paragraph of the identification division specifies that the mode of arithmetic for the source unit is standard arithmetic.

D.17.3.2 Examples

These examples demonstrate a few of the features and requirements of standard-binary arithmetic.

In the following examples, "sbidi" stands for standard-binary intermediate data item, "sdidi" stands for standard-decimal intermediate data item, "ulp" is an abbreviation for "unit in last position", and "ir-n" stands for the "nth intermediate result".

"Rounding" in the examples below implies the application of the MODE phrase of the INTERMEDIATE ROUNDING clause according to the rules of the floating-point standard. The application of the ROUNDED clause is explicitly described in the examples.

```
1  A  pic s9(4) value +8000.
1  B  pic s9(4) value +3000.
1  C  pic s9(4) value –4001.
1  D  pic s9(4) value is zero.
1  E  pic s99V99 value is +56.79.
1  F  usage float-binary-34.
1  G  usage float-decimal-34.

COMPUTE D = A + B + C
    *> begin
    *> convert A to sbidi  as ir-1
    *> convert B to sbidi  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> convert C to sbidi  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> Convert ir-1 to sdidi format with rounding as ir-2
    *> move ir-2 to D (truncating fractional digits)
    *> end

COMPUTE D ROUNDED = D + E
    *> begin
    *> convert D to sbidi  as ir-1
    *> convert E to sbidi  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> convert ir-1 to sdidi format with rounding as ir-2
    *> align decimal point of ir-2 to that of D
    *> adjust ulp of ir-2 according to DEFAULT ROUNDED MODE clause
    *> move adjusted ir-2 to D
    *> end

COMPUTE F = A + B + C
    *> begin
    *> convert A to sbidi  as ir-1
    *> convert B to sbidi  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> convert C to sbidi  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> move ir-1 to F (ir-2 and F are in the same format)
    *> end

COMPUTE G ROUNDED = G + E
    *> begin
    *> convert G from sdidi to sbidi format  as ir-1
    *> convert E to sbidi format  as ir-2
    *> add ir-1, ir-2 with rounding, store in ir-1
    *> convert ir-1 from sbidi to sdidi format  as ir-2
    *> NOTE no rounding applies; ir-2 and G are in the same format
```

```
        *> move ir-1 to G
        *> end
```

D.17.4 Maximum error due to truncation and rounding

This discussion presents the maximum error due to truncation of low-order digits and rounding for primitive operations involving one or two operands. The abbreviation "ulp" means unit in the last position. In general, analysis of a specific situation may reduce the maximum error expected from truncation or rounding; or may eliminate it completely.

D.17.4.1 Arithmetic statements

For the COMPUTE statement, see the paragraph on arithmetic operations in arithmetic expressions below.

D.17.4.1.1 ADD and SUBTRACT statements

Given the following examples:

```
ADD A TO B
ADD C TO D GIVING E
SUBTRACT A FROM B
SUBTRACT C FROM D GIVING E
```

if the composite of operands, the pair A and B or the pair C and D, has fewer than 32 digits and if the destination, B and E, respectively, has at least as many places to the right of the decimal point as the composite of operands, then there is no error due to truncation or rounding. In all other cases, if ROUNDED is specified, the maximum error due to rounding is 1/2 ulp of the destination and if ROUNDED is not specified the maximum error due to truncation is 1 ulp of the destination.

D.17.4.1.2 MULTIPLY statement

Given the following examples:

```
MULTIPLY A BY B
MULTIPLY C BY D GIVING E
```

if the sum of the number of digits in the source operands, the pair A and B or the pair C and D, is less than 32 and if the destination, B and E, respectively, has at least as many places to the right of the decimal point as the sum of the digits to the right of the decimal point of the source operands, then there is no error due to truncation or rounding. In all other cases, if ROUNDED is specified, the maximum error due to rounding is 1/2 ulp of the destination and if ROUNDED is not specified the maximum error due to truncation is 1 ulp of the destination.

D.17.4.1.3 DIVIDE statement

Given the following examples:

```
DIVIDE A INTO C
DIVIDE A INTO B GIVING C
DIVIDE B BY A GIVING C
DIVIDE A INTO B GIVING C REMAINDER D
DIVIDE B BY A GIVING C REMAINDER D
```

the maximum error due to truncation or rounding of the quotient C or remainder D can exceed 1 ulp of the destination for both the quotient and the remainder.

D.17.4.2 Arithmetic operations in arithmetic expressions

In general, the maximum error due to truncation for each primitive operation is 1 ulp of the standard intermediate data item. However, analysis of specific situations may identify cases where the maximum error is less or there is no such error. Rounding does not apply.

D.17.4.2.1 Unary operations and operands with no operator

Given the following examples:

```
+A
–A
 A
```

there is no loss due to truncation.

D.17.4.2.2 Addition and subtraction

Given the following examples:

```
A + B
A – B
```

If the composite of operands A and B has fewer than 33 digits, there is no loss due to truncation.

D.17.4.2.3 Multiplication

Given the following example:

```
A * B
```

If the sum of the number of significant digits in A and B is fewer than 33, there is no loss due to truncation.

D.17.4.2.4 Division

Given the following example:

```
A / B
```

There is no way to predict a reduced error from truncation by examining the number of digits in either the dividend A or the divisor B.

D.17.4.2.5 Exponentiation

Given the following example:

```
A ** B
```

There is no loss due to truncation in the following cases:

- B has a value of zero;
- B has a value of one;
- B has a value of two and A has fewer than 17 significant digits;
- B has a value of three and A has fewer than 11 significant digits;
- B has a value of four and A has fewer than 9 significant digits.

D.17.4.2.6 SQRT function

Given the following example:

```
Function SQRT (A)
```

There is no way to predict a reduced error from truncation by examining the number of digits.

D.17.5 Examples

These examples demonstrate a few of the features and requirements of standard arithmetic. In order to make the examples easier to follow, the maximum number of digits in a data division data item is assumed to be 4 and the standard intermediate data item contains 5 significant digits. The results before truncation and normalization are only one possible format and do not necessarily correspond to any real implementation, however the algebraic value after truncation is what is required.

In the following examples, "sidi" stands for "standard intermediate data item" and "ir-n" stands for the "nth intermediate result". The notation used for the standard intermediate data items and intermediate results is scientific notation. For example "+1.10000 E+4" and "+ .11000 E+5" are both representations of eleven thousand and "+ .06999 E+5" and "+ .69990 E+4" both represent six thousand nine hundred ninety-nine.

```
1 A pic s9(4) value +8000.
1 B pic s9(4) value +3000.
1 C pic s9(4) value –4001.
1 D pic s9(4) value is zero.
1 E pic s99v99 value +56.79

COMPUTE D = A + B + C.

begin
calculate ir-1 = A + B
    convert A to sidi                                    +8.0000 E+3
    convert B to sidi                                    +3.0000 E+3
    add (one possible representation)                   +11.0000 E+3
    normalize and truncate, store in ir-1                +1.1000 E+4
calculate ir-2 = ir-1 + C
    no conversion needed for ir-1                        +1.1000 E+4
    convert C to sidi                                    –4.0010 E+3
    add (one possible representation)                    +0.6999 E+4
    normalize (truncation not needed), store in ir-2     +6.9990 E+3

store ir-2 in D
    round from last                                      +6.9990 E+3
    move, D =                                       +6999
end

COMPUTE D ROUNDED = D + E

begin
calculate ir-1 = D + E
    convert D to sidi                                    +6.9990 E+3
    convert E to sidi                                    +5.6790 E+1
    add (one possible representation)                    +7.05579 E+3
   (normalize not needed) truncate, store in ir-1        +7.0557 E+3

store ir-1 in D
   (Because of ROUNDED, the sidi is not rounded.
    However, since D has the maximum number of
    digits, the effect is the same.)
    apply ROUNDED
        align                                       + 7055.7
        round causes increase in ulp                + 7056
        move, D =                                   + 7056
end

1 E pic S9(4) value +1291.
1 F pic S9(3) value +569.

COMPUTE E = E * F

begin
calculate ir-1 = E * F
    convert E to sidi                                    +1.2910 E+3
    convert F to sidi                                    +5.6900 E+2
    multiply (one possible representation)               +0.734579 E+6
    normalize, truncate, store in ir-1                   +7.3457 E+5
```

```
store ir-1 in E
        round from last                                  +7.346 E+5
        move, SIZE ERROR
end

COMPUTE E = F * 0.754

begin
calculate ir-1 = F * 0.754
    convert F to sidi                                    +5.6900 E+2
    convert 0.754 to sidi                                +7.5400 E-1
    multiply (one possible representation)               +4.29026 E+2
    (normalize not needed) truncate, store in ir-1       +4.2902 E+2

store ir-1 in F
        round from last                                    +4.290 E+2
        move, E =                                        +429
end
```

D.18 Object oriented concepts

Object oriented programming is all about developing and implementing application systems as sets of interacting software objects.

A software object, like most objects in everyday life, such as an automobile, has a unique identity, and certain attributes and behaviors. The automobile has a unique identity, its serial number; it has many attributes such as color, number of doors, and weight. It also has such behaviors as forward, reverse, accelerate, shift, and the like. Software objects are used to model real world objects and as such they abstract the key concepts of the real world object in software. A software object used to model an automobile would for example, have a unique identity, and attributes such as color, weight, and length, as well as such behaviors as forward and reverse.

Software objects can be used to model any of the concepts germane to a given problem domain. For example they can represent bank accounts, employees, parts, processes, programs, fields, files, structures and the like.

Therefore, we can say a software object is an entity that has a unique identity, specific data values, and specific behaviors or program code. The program code is organized into small modules. In object oriented terminology these modules are called methods. Data is encapsulated within each object and can only be accessed by using one or more of the object's methods.

D.18.1 Classes

To facilitate dealing with the hundreds or even thousands of different software objects that can exist in an application system, objects are organized into classes. A class is a group of objects that have a common data structure and that all use the same methods. This means that the data structure can be defined for the class. Each object within the class has a unique set of data values that correspond to the class structure. It also means that the methods are defined once at the class level and are used by each of the objects of the class.

Example:

A banking application will require many individual accounts. Each account will have data associated with it, for example account-balance and date-opened. Each account will have methods that allow other parts of the application to access an individual account such as deposit, withdraw and current-balance. The checking account class defines the data layout and methods for all of the individual checking accounts, thus they all work in the same way and serve the same purpose.

D.18.2 Objects

Every object belongs to exactly one class; there may be zero, one, or more objects of any given class.

D.18.2.1 Object instantiations

The individual members of a class are called instance objects, or simply instances. For example, an individual checking account object can be called a checking account object instance, or a checking account instance. We will use the term instance to refer to an individual member of a class.

All instances of a class share the same data definition, but each instance has its own unique values.

The instances of a class also share the same methods. A method is defined once for the class, but each instance behaves as if it is the sole owner of the methods defined for the class. The methods are shared and can be used by all instances. This facility of object oriented programming environments permits the data for each instance to be hidden (encapsulated) because it can be accessed only via the methods of the class.

The conventions used in all of the code fragments and the sample bank application are as follows:

— Class names and interface names use camel case, the first letter of each word is capitalized. For example, the class checking account is "CheckingAccount".

— Method names use lower case for the first word and camel case for subsequent words, for example, the method deposit is "deposit"; and the method calculate charges is "calculateCharges".

— Data item names always consist of at least two lower case words separated by a "-", for example, customer name is "customer-name", an object is "an-object", and so forth.

Example:

Each checking account is represented by an account instance. Each instance has its own copy of the data described by the class, the customer's name, the current balance, and the date opened. Each instance uses the methods defined for the class to carry out its functions, for example, the deposit method credits the account.

Within the body of program code that defines a class, instances of the checking account class could be defined as follows:

```
. ...
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  checking-account.
   03  customer-name      PIC X(35).
   03  current-balance    PIC S9(9)V99.
   03  date-opened        PIC  9(8).
    ...
PROCEDURE DIVISION.
METHOD-ID.   deposit.
   method code
END METHOD deposit.
. ...
METHOD-ID.    withdraw.
   method code
END METHOD withdraw.
. .
END OBJECT.
```

D.18.2.2 Object data definitions

Since each instance object is unique, it has a unique reference value that is generated by the runtime system when the instance is created. An object's reference value serves as a pointer to that specific instance. An object reference value can be thought of as a key that identifies a specific instance. The usage clause provides a means to define a data item, called an object reference, to hold an object's reference value. In the following example, data items a-checking-account and an-account are object references:

Example:

```
. . .
```

```
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  checking-account.
    03  customer-name          PIC X(35).
    03  current-balance        PIC S9(9)V99.
    03  date-opened            PIC 9(8).
01  a-checking-account         USAGE IS OBJECT REFERENCE.
01  an-account                 USAGE IS OBJECT REFERENCE.
PROCEDURE DIVISION.
....
```

The data item a-checking-account can be used to refer to a specific instance. A data item that has been defined as an object reference can be set equal to another data item defined as an object reference through the use of the SET statement as follows:

```
    SET an-account TO a-checking-account.
```

The above statement will transfer the value of the object reference in a-checking-account to an-account.

D.18.2.3 Object references

An object reference is a data item that contains a reference to an object. The content of the object reference is used to locate the object and information associated with the object.

An application may ensure at compile time that an object of one class, say employee, will never be used as an object of an unrelated class, say account. This is done by using an object reference described with either a class-name, an interface-name, or an ACTIVE-CLASS phrase. If a class-name is specified, the data item can only be used to reference an object of the class specified, or one of its subclasses, as discussed in D.18.4.1, Inheritance. If an interface-name is specified, the data item can only be used to reference an object described with an IMPLEMENTS clause that references the interface specified, as discussed in D.18.4.4.1, Class polymorphism. If the ACTIVE-CLASS phrase is specified, the data item can only be used to reference an object of the same class as that of the object with which the method was invoked. Object references for ACTIVE-CLASS are of special significance as returning items. This capability is needed for defining a New method in the BASE class that works as documented without violating the conformance rules, and it allows writing user methods that do object creation in conformance with the definition of the class hierarchy.

Alternatively, an application may use a universal object reference, one that can refer to any object. A data item is defined as a universal object reference by omitting all optional phrases from the OBJECT REFERENCE phrase of the USAGE clause. Runtime validation may be used to ensure that an object has the correct interface as described in 14.8, Conformance for parameters and returning items, but this approach does require more runtime resources.

Examples:

The definition of a data item that can refer only to an object of the CheckingAccount class or one of its subclasses is:

```
1  an-account  USAGE IS OBJECT REFERENCE CheckingAccount.
```

The definition of a data item that can hold a reference to any object is as follows:

```
1  an-object  USAGE IS OBJECT REFERENCE.
```

D.18.2.4 Factory objects

As stated previously, a class describes the data for each instance of the class and defines the methods that can be used by each instance of the class. Each class has one object, called the factory object, that is responsible for functions, such as creating a new instance of the class and managing data associated with all instances of the class.

A factory object can be thought of as an instance of a special kind of class and has data (factory data) and methods (factory methods). The data and methods for the factory object are defined as part of the class definition.

Every instance of a class is created by the factory object of that class. When an object is created, the data descriptions in the class are used to allocate storage for the instance.

Example:

The checking account class shown below describes a factory object, which is called the checking account factory object. To create a new checking account instance, a method in the checking account factory object is used. To keep track of the number of checking account instances a data item in the factory object can be used. Whenever a new instance is created, 1 can be added to the value; whenever an instance is removed, 1 can be subtracted from the value.

Sample code for the checking account factory object could be as follows:

```
CLASS-ID. CheckingAccount INHERITS Base.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
   CLASS Base.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  number-of-accounts        PIC 9(5).
PROCEDURE DIVISION.
METHOD-ID.  newAccount.
create-account.
   add 1 to number-of-accounts.
. ...
END METHOD newAccount.
END FACTORY
OBJECT.
. ...
END CLASS CheckingAccount.
```

D.18.3 Methods

A method is procedural code that defines a specific function required by all of the instances of a class. A method may be thought of as a module or subroutine. A class may define as many methods as it needs to manage the data defined for the class. Methods are typically only a few lines of procedural code, but may be as many lines as required to accomplish a specific function.

An instance object is used by invoking one of its methods. This facility is similar to the call facility. With conventional coding techniques, one program activates another program by issuing a call. With object oriented programming techniques, one object activates another object by issuing an invoke.

Methods are distinguished by their name alone. A class cannot define two methods with the same name. If a subclass defines a method with the same name as a method in an inherited class, the method in the inheriting class overrides the inherited method.

D.18.3.1 Method invocation

Any program or method can invoke a method to act on an object. The name of the method specified on the invocation statement will be the method executed. The invocation statement also allows arguments to be passed to the method and also allows the method to return a result.

Example:

Whenever an application needs to use an object, it invokes a method to act on the instance object. Let's assume the CheckingAccount class contains the methods deposit, withdraw and balance and that an-account references an instance of the Account class. The syntax to deposit an amount to an account is as follows:

```
        INVOKE an-account "deposit" USING in-amount
```

Similarly, the syntax to determine the current balance of an account is:

```
       INVOKE an-account "balance" RETURNING current-balance
```

An equivalent statement illustrating inline method invocation is:

```
       MOVE an-account::"balance" TO current-balance
```

When the application needs to determine the balance of a specific account, a conventional program or a method will request the instance to activate its balance method. Code fragments to accomplish this are shown below:

Assume a program wants to determine the balance of a checking account.

Program Code

```
WORKING-STORAGE.
....
01  a-checking-account-object USAGE IS OBJECT REFERENCE CheckingAccount
....
77  the-balance               PIC S9(8)V99 VALUE ZERO.
....
PROCEDURE DIVISION.
....
INVOKE a-checking-account-object "balance"  RETURNING the-balance.  *> assume the object
                                             *> referenced by a-checking-account-object
                                             *> contains the reference to the desired
                                             *> account
```

Checking Account Class

```
....
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  checking-account.
    03  customer-name      PIC X(35).
    03  current-balance    PIC S9(9)V99.
    03  date-opened        PIC  9(8).
....
PROCEDURE DIVISION.
....
METHOD-ID.    balance.
DATA DIVISION.
....
LINKAGE SECTION.
01  ls-balance             PIC S9(8)V99.
....
PROCEDURE DIVISION RETURNING ls-balance.
return-balance.
    MOVE current-balance TO ls-balance.
    EXIT METHOD.
END METHOD balance.
....
```

D.18.3.2 Method prototypes

Method prototypes are method skeletons that define the method name, parameters, and information necessary to describe those parameters such as that specified in the REPOSITORY paragraph and the SPECIAL-NAMES paragraph. They do not include the procedural code. In essence they provide all of the information required to invoke a method. Method prototypes are used to specify interfaces and are specified in interface definitions.

Example:

The method prototype for the calculateInterest method of the SavingsAccount class is as follows:

```
METHOD-ID.  calculateInterest.
DATA DIVISION.
LINKAGE SECTION.
01  interest-rate          PIC S9(3)v9999.
```



```
01  interest-amount        PIC  S9(9)V99.
PROCEDURE DIVISION USING interest-rate RETURNING interest-amount.
END METHOD calculateInterest.
```

D.18.4 Other object oriented programming features

Some of the other capabilities are Inheritance, Interfaces and Polymorphism, and Conformance.

D.18.4.1 Inheritance

One of the language features that separates object oriented languages from conventional programming languages is the ability to develop a hierarchy of classes, as shown by the example in figure C.4, Manager class.

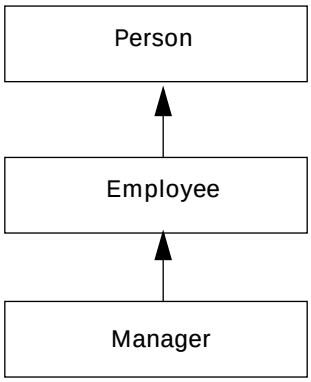


Figure C.4 — Manager class

The manager class is a subclass of the employee class which in turn is a subclass of the person class. Or said another way, the employee class is the superclass of the manager class and the person class is the superclass of the employee class. At any point in the hierarchy, the classes above a given class are its superclasses or its ancestors and the classes below are its subclasses or its children. A subclass includes all of the capabilities of all of its ancestors and additionally may add to or override these capabilities. For example, the methods of the person and employee classes are available to an instance of the manager class as well as the methods defined for the manager class.

Inheritance is the mechanism used to develop class hierarchies.

Inheritance supports a hierarchy of classes, where every instance of a subclass can be used "as if it were" an instance of its superclasses. For example, a manager object could be used as if it were an employee object, and an employee object could be used as if it were a person object. This is the principle of conformance between classes. When classes conform, a data item declared as a reference to an object of a given class may in fact reference an object of any class that descends from that given class.

Inheritance represents an "is a" relationship between two classes and is a way of specializing a higher level class. In figure C.4, Manager class, a manager "is an" employee and an employee "is a" person. All the object data definitions described in the superclasses, person and employee, plus the object data definitions for the class itself, manager, are used to create an instance of the manager class. Also, the methods defined by the superclasses are inherited by the subclass and are used to directly operate on any instance of manager.

Both factory and instance data and methods of all ancestor classes are inherited by a class that inherits from another class.

When inheritance is used to define a subclass, data in the superclass is encapsulated because methods defined for the subclass are not allowed to directly access the data items defined for the superclasses. It requires all subclasses to use methods defined for the superclasses to access the data items defined for the superclasses. As an example, if the employee class defined the data item "employee-name", the employee class would have to include a method, say getName, to allow any subclass to retrieve the employee name.

A class may inherit from more than one other class, this is called multiple inheritance.

Example of single inheritance:

A bank will have different kinds of accounts, and yet they are all accounts. If we consider checking accounts and savings accounts, they have a number of common features, they both have an owner and a balance. They also have some different features, the fact that checks are allowed for one and the other pays interest. It makes sense to have a basic account class that contains the common parts and then to use inheritance to define the checking account and the savings account subclasses. Thus the account class defines what is common to all accounts; the checking account class defines what is specific to checking accounts; and the savings account class defines only what is specific to savings accounts. Any changes to the account class will be picked up by the inheriting classes automatically. These relationships can be represented as shown in figure C.5, Banking hierarchy.

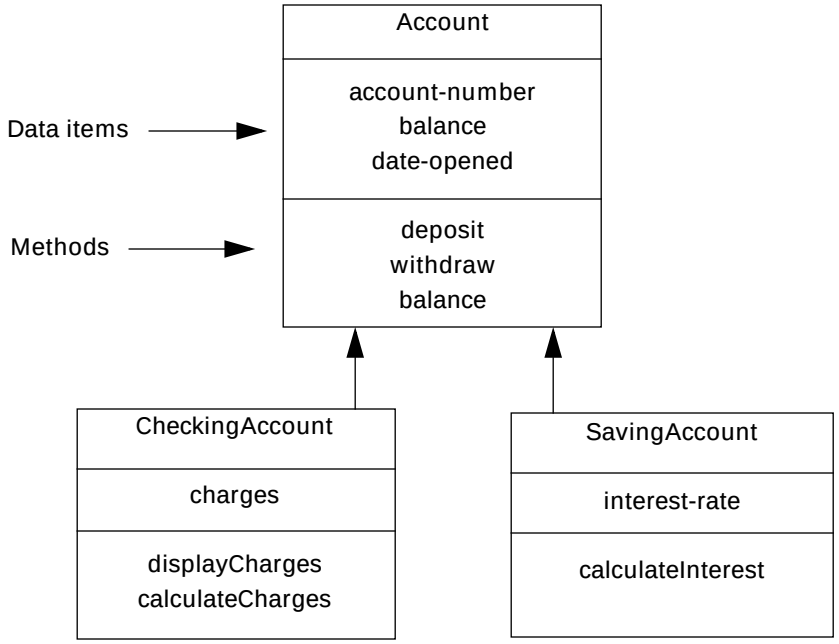


Figure C.5 — Banking hierarchy

In the example shown, each instance of CheckingAccount is automatically created with memory allocated for the attributes account-number, balance, date-opened and charges. Additionally, the methods deposit, withdraw, and balance inherited from Account and the methods displayCharges and calculateCharges defined in the CheckingAccount class can act on each instance. Each instance of SavingsAccount is automatically created with memory allocated for the attributes account-number, balance, date-opened and interest-rate. Each instance of SavingsAccount can access the methods deposit, withdraw and balance inherited from Account and the method calculateInterest defined for itself.

Some sample code for the account and checking account classes is shown below:

Account Class

```
CLASS-ID.  Account INHERITS Base.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY
   CLASS Base.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  number-of-accounts        PIC 9(5).
PROCEDURE DIVISION.
METHOD-ID.   newAccount.
   method code
END METHOD newAccount.
....
END FACTORY.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  ACCOUNT-INFORMATION.
   03  account-number         PIC X(12).
   03  balance                PIC S9(8)V99.
   03  date-opened            PIC 9(8).
. ...
PROCEDURE DIVISION.
METHOD-ID.   deposit.
....
END METHOD deposit.
METHOD-ID.   withdraw.
....
END METHOD withdraw.
METHOD-ID.   balance.
....
END METHOD balance.
....
END CLASS Account.
```

CheckingAccount Class

```
CLASS-ID. CheckingAccount INHERITS Account.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS Account.
....
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  checking-account
   03  charges            PIC S9(8)V99.
. ...
PROCEDURE DIVISION.
METHOD-ID.    displayCharges.
....
END METHOD displayCharges.
METHOD-ID.    calculateCharges.
....
END METHOD calculateCharges.
....
END CLASS   CheckingAccount.
```

D.18.4.2 Restricting inheritance and modification with the FINAL clause

It may be desired that no extension be done to a class through inheritance. This, for example, might be needed by library providers who want to control the implementation of a class. To provide this functionality, a class may be declared 'final' if its definition is complete and no subclasses are desired or required. A compile-time error occurs

if the name of a final class appears in the INHERITS clause of another class declaration; this implies that a 'final' class cannot have any subclasses.

Because a final class never has any subclasses, the methods of a final class are never overridden. This may be an overkill in some cases, and it might be desired that only a few methods not be overridden. For that purpose, we can use the attribute 'final' with a method of any class, to prohibit the subclasses of that class from overriding that method. This attribute can also be used redundantly with the methods of a final class.

Restricting methods from being overridden also helps in 'pairing' of methods of a class. Here's an example: Suppose a class A defines a method 'bar' calling another method 'foo' defined in the same class. If there is a subclass B that also defines 'foo', and if we invoke 'bar' on an object of class B, the method 'bar' (inherited from class A) will end up calling the function 'foo' defined for B and not the original 'foo'. If however, we specify the FINAL clause with the function 'foo' in class A, it cannot be overridden in class B, and the programmer can ensure that 'bar' will always invoke the same 'foo'.

The FINAL attribute has to be handled carefully while dealing with multiple inheritance. If two classes A and B define a method of the same name, and if a class C inherits from both of them, the method definitions are not allowed to have the FINAL clause specified. However, if the same method is inherited through two classes which had the same superclass defining that method, the method is allowed to have the FINAL clause specified in its definition. This would happen for a diamond shaped multiple inheritance, where classes B and C inherit from a class A, and then class D inherits from both B and C. Class A can have methods with the FINAL clause specified, and though D will appear to inherit two methods of the same name with the FINAL clause, it is acceptable as they are the same method implementations.

D.18.4.3 Conformance

Conformance allows the compiler to check the application code and determine if any class hierarchy inconsistencies can occur at runtime. A data item can be constrained to be an object reference for only objects of a specific class or its subclasses by inserting the class name after the USAGE IS OBJECT REFERENCE clause. Additionally, the compiler can determine if the arguments passed to a method match the parameters specified in the USING phrase of the procedure division header. Arguments must match the parameters exactly to ensure conformance.

Example:

One part of a banking application can be written to deal with any kind of account. Data items that have been declared such as:

```
01  an-account          USAGE OBJECT REFERENCE Account.
```

can reference any object of the account class or any object of a class that inherits from the account class. The rules of conformance ensure that the subclasses of the account class can be used in exactly the same way as objects of the account class itself. The underlying implementation of a subclass may be different from the original account class, but the interface is guaranteed to be compatible.

A different part of the banking application can be written to deal only with a specific kind of account. For example, the data item an-account would be defined as follows to hold a reference to a checking account object or to any object of a class that inherits from the CheckingAccount class. Note that in this example, no classes inherit from the CheckingAccount class.

```
01  an-account      USAGE OBJECT REFERENCE CheckingAccount.
```

If a source element contains code that attempts to put a reference to an Account object into the data item declared to contain a checking account object, the compiler will warn the user that there is a potential error. It should be noted that the compiler cannot necessarily determine that this actually is an error, only that it might be an error. Additionally, the compiler can check to ensure that the arguments and parameters match.

The policy for conformance checking is conservative and errs on the side of caution.

These are some examples of restrictions imposed by compile time conformance checking, even though at runtime a conformance violation might not actually exist:

1) Let's assume there is a class A with a subclass A1, and a source element containing the following definitions:

```
1 or-1 object reference A.
1 or-2 object reference A1.
```

The statement

```
SET or-2 to or-1
```

is invalid, because or-1 may contain, for example, a reference to class A, which is not valid in or-2. At runtime, or-1 might actually contain a reference to A1, which would be a valid content of or-2. This is, however, not predictable at compile time. Therefore, the SET rules require that the class of the sending operand, A in this case, is the same class or a subclass of the class of the receiving operand (A1), which is not the case.

2) It is not permitted to pass or to return a strongly-typed group item having a subordinate item that is an object reference for ACTIVE-CLASS. At runtime, the class of the activating element may be different from the class of the activated element; that is, the two object references are restricted to different classes. These, again, are not known at compile time. (An object reference cannot be subordinate to a group item that is not strongly typed.)

```
Class A.
...
Method-Id.  M-A.
...
1 or-1 object reference b.
1 t-a typedef strong.
     2 ...
    2 or-a object reference active-class.
1 a-a type t-a.
...
Procedure Division.
...
    Invoke B "New" returning or-1
    Set or-a to self
    Invoke or-1 "M-B" using a-a *> Invalid argument
...
End Method M-A.
End Class A.

Class B.
...
Method M-B.
...
Working-Storage Section.
...
1 t-a typedef strong.
    2 ...
    2 or-b object reference active-class.
...
Linkage Section.
...
1 a-b type t-a.
...
Procedure Division using a-b.
...
Exit method M-B.
End Method M-B.
End Class B.
```

In this invalid example, or-a is restricted to class A, or-b to class B, and normally any valid content of or-a will be invalid for or-b, and vice-versa.

3) Although returning an object reference for ACTIVE-CLASS is generally allowed, there are still restrictions due to the compile-time checking requirement. The compiler does not know the object that will be used to invoke

the method; it does, however, know the object reference that is used to invoke the method containing the object reference to be returned. It is possible for the compiler to derive some information about the item to be returned.

Consider the following class:

```
Class-id. C inherits B.
Factory.
  Method-id. M
  Linkage section.
  01 or-1 object reference active-class.
  Procedure division returning or-1.
  End method M.

End class C.
```

Consider:

```
Invoke C "M" returning anObj.
```

The compiler knows that the object returned by method M is of class C or some subclass of C. This knowledge can be used to detect some errors. For example, suppose we have this class hierarchy:

```
B
|
C
|
D
```

Consider the following statements:

```
01 or-B object reference B.
01 or-C object reference C.
01 or-D object reference D.

Invoke C "M" returning or-B
Invoke C "M" returning or-C
Invoke C "M" returning or-D
```

The compiler can statically determine that the first and second invokes are valid but the 3rd invoke is invalid, since it might result in storing an object of type C in an object reference of type D.

4) In principle, method invocation on a specific object identified at runtime is permitted for any method that is defined for that object. Compile time checking, however, restricts the eligible methods to those that are known at compile time. For example, if the specified object reference is restricted to a specific class, a method with this name must be defined in the class specified in the object reference. Thus it is possible to invoke an overriding method defined in a subclass of the specified class, but not a method that is defined only in the subclass, but not in the parent class.

Let C-1 be a class with a subclass C-2, where C-1 contains a method M-1 and C-2 contains a method

M-2. Assume further a client program or method contains:

```
1 or-1 object reference C-1.
   Invoke or-1 "M-1"
```

This is valid. Even when or-1 actually references an object of C-2, there is no problem, because it is still the same M-1 in class C-1. Also, there is no problem when M-1 is defined in C-2 as a method overriding the M-1 of C-1, because the signature of the overriding M-1 is still the same.

But:

```
Invoke or-1 "M-2"
```


is invalid, even when or-1 actually references an object of C-2, because the signature of M-2 is not known at compile time, and there is no way of conformance checking at compile time.

Note, however, that the object modifier can be used to get type-safe access to M2, with conformance checking at runtime:

```
Invoke or-1 as C2 "M-2".
```

D.18.4.4 Polymorphism

Polymorphism is supported by COBOL in two different ways. Class polymorphism is supported through class inheritance and the use of interfaces. Parametric polymorphism is supported through method overloading.

D.18.4.4.1 Class polymorphism

Class polymorphism is generally provided through the class inheritance. In COBOL, the use of interfaces also provides class polymorphism.

An interface definition defines a subset of methods of any class implementing that interface. It provides a view of the methods that can be invoked for the class, including the names and parameter specifications for each method. That is, only method prototypes are described in the source unit of an interface definition.

A class may implement several interfaces. Each interface may include one or more of the methods of that class.

An object that implements all of the methods defined in an interface conforms to that interface. The application class hierarchy forms a hierarchy of conforming interfaces.

Example:

A banking application may have defined a method in the Account class that prints the data values associated with each instance, for example, current owner and balance. Likewise, a method in the Customer class can print the name and address of the customers it represents. If there is a need for a generalized routine that prints things, with correct page formatting, an interface can be defined that contains the methods associated with printing. Any object that implements this interface can then be printed by this routine. This illustrates polymorphism.

Sample code for the Print Interface is shown below:

```
INTERFACE-ID. PrintReport.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
PROCEDURE DIVISION.
METHOD-ID.   printRpt.
END METHOD printRpt.
. ...
. ...
END INTERFACE PrintReport.
```

D.18.4.4.2 Parametric polymorphism

Method overloading provides the ability to declare two or more methods of the same name, but with a different number of parameters and types. Each method has a method resolution signature that consists of the method name and all of the relevant information from the definition of each of the parameters, and from the returning item. During the resolution of the method, the signature derived from the invocation is compared with all methods of the same name. If the signatures match exactly, that method is bound. If not, the method that most closely matches the signature while still conforming is bound.

No two methods within a class may have the same signature.

Example:

Suppose we have three methods that print. The first method invokes the print method of whatever object pointer is passed to it. The second formats a packed decimal field and prints it, the third prints a variable length character string.

```
Method-id. PrintIt.
Linkage section.
01  In-o  usage object.
Procedure division using In-o.
    invoke in-o "PrintMe".
End Method PrintIt.

Method-id. PrintIt.
Working-Storage section.
01  Out-p pic ZZZ,ZZZ,ZZZ.
Linkage section.
01  In-p  BINARY-SHORT.
Procedure division using In-p.
    move In-p to Out-p.
    display out-p.
End Method PrintIt.

Method-id. PrintIt.
Working-Storage section.
01  Out-p pic $$$,$$$.99.
Linkage section.
01  In-p  pic s9(5)v99 packed-decimal.
Procedure division using In-p.
    move In-p to Out-p.
    display out-p.
End Method PrintIt.

Method-id. PrintIt.
Linkage section.
01  In-x pic x(20).
01  In-len pic 9(4).
Procedure division using by value In-x, In-len.
    display In-x(1:In-len).
End Method PrintIt.
```

If we invoke PrintIt with an object reference as in

```
Invoke aClass "PrintIt" using anObject.
```

we would invoke the first method, which would invoke the "PrintMe" method of anObject. If we invoked PrintIt as follows:

```
Invoke aClass "PrintIt" using "FooBar", 3.
```

we would invoke the last method, and display "Foo".

Invoking Printit with a numeric as follows:

```
Invoke aClass "PrintIt" using 32
```

could match either the second or the third method, since the literal "32" is treated as USAGE DISPLAY. When there are two methods that match equally well, the first of the methods that match is the method to which we will resolve. In this case, it would be the second method, and we would display "32". Placing the method that is preferred first in the compilation unit assures that the method will be selected when the choice is ambiguous.

D.18.5 Object management

D.18.5.1 Objects

As mentioned previously, objects are allocated at runtime and are accessed only through object references. An object is created by invoking a method in a factory object. The object continues to exist until it can no longer be accessed. The object can no longer be accessed when no object reference data item references that object.

D.18.6 Class library

A small class library is provided that can be used, extended, or integrated into business solutions. The BASE class is the single top node of the class library and includes methods that support object creation and initialization, and also other primitive functions useful for objects.

D.18.7 Parameterized classes

A parameterized class is a skeleton class definition that when passed the appropriate parameters will create a new class tailored to perform a given function. Parameterized classes allow developers to create classes that have common behavior. For example, a single parameterized collection class can be used to define many types of collection classes.

Example:

Let's consider container classes which are typically used to hold references to objects of a given class. In a banking application, a specific type of container class, say AccountCollection, could be used to hold the list of identifiers to each account object. To generate a collection class that is capable of holding or managing only identifiers of checking account objects, the class name CheckingAccount is specified as a parameter when the collection class is created.

Suppose AccountCollection is a parameterized class whose definition is given elsewhere. Part of its class definition is as follows:

```
CLASS-ID.  AccountCollection INHERITS Base USING X.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS Base
CLASS X.
...
END CLASS AccountCollection.
```

In another source unit, the following would create a class which is a collection of checking accounts and an object reference for this new kind of class.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS CheckingAccountCollection EXPANDS AccountCollection USING CheckingAccount.
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  a-checking-account-collection USAGE OBJECT REFERENCE CheckingAccountCollection.
...
```

D.18.8 Files in object orientation

Files in object oriented applications can be specified in two different ways:

1) Files in Instance Objects

2) Files in Factory objects

D.18.8.1 Files in instance objects

A file specified in an instance object means that the instance definition contains the FILE-CONTROL paragraph and the file section. One or more instance methods will contain the file processing statements such as OPEN, CLOSE, READ and WRITE. All of the instance methods have visibility to the records associated with the file.

When a class that contains a file specified in the instance definition is inherited, each direct or indirect descendent also inherits the file specification. The instance objects of each of these subclasses have their own file connector unless the EXTERNAL clause is specified for the file. Dynamic file assignment or file sharing may be used to resolve conflicts in accessing the physical files associated with these file connectors.

Specifying dynamic file assignment in the file control entry permits a class to be used to define a logical file of a given structure, and each instance can associate a different physical file with its own file connector and perform I-O on that physical file.

Sample code for dynamic file assignment is illustrated below. Note that the MOVE statements only have an effect on the dynamic assignment when a subsequent OPEN statement for the file connector is executed.

```
CLASS-ID. Employee INHERITS Base.
        ...
 OBJECT.
       ...
 FILE-CONTROL.
      SELECT EMPLOYEE-FILE ASSIGN USING FILE-REF.
 DATA DIVISION.
 FILE SECTION.
 FD  EMPLOYEE-FILE
      ...
 WORKING-STORAGE SECTION.
 01  FILE-REF  PIC X(16) VALUE SPACES.
      ...
 PROCEDURE DIVISION.

 METHOD-ID. readFile.
      ...
 WORKING-STORAGE SECTION.
 01  EMPLRCD.
     03  SSN                        PIC 9(9).
     03  NAME  ...
 PROCEDURE DIVISION.
    ...
    MOVE 'external-ref01' TO FILE-REF
    OPEN  INPUT EMPLOYEE-FILE
    ...
    READ EMPLOYEE-FILE NEXT RECORD INTO EMPLRCD
    CLOSE EMPLOYEE-FILE
    ...
    MOVE 'external-ref02' TO FILE-REF
    OPEN  INPUT EMPLOYEE-FILE
    ...
    READ EMPLOYEE-FILE NEXT RECORD INTO EMPLRCD
    ...
```

D.18.8.2 Files in factory objects

When a file is specified in a factory, this means that the factory definition contains the FILE-CONTROL paragraph and FILE SECTION. One or more of the factory methods will contain the file processing statements such as OPEN, CLOSE, READ and WRITE. All of the factory methods have visibility to the data on the file without the use of the EXTERNAL clause.

Inherited Factory Object Definitions - It is important to note that when a class that contains a file specified in the factory object definition is inherited, each direct or indirect descendent also inherits the file specification. As in the case of inherited object definition above, the factory objects of each of these subclasses have their own file connector unless the EXTERNAL clause is specified for the file. Dynamic file assignment or file sharing may be used to resolve conflicts in accessing the physical files associated with these file connectors.

D.18.9 Exception objects

Exception objects are used similarly to predefined exception conditions associated with exception-names. They provide, however, more flexibility and allow access to a wider variety of information for resolving exception situations, compared to the predefined or user-defined exception conditions, especially when used in object-oriented applications.

Although in principle any object of any class could serve as an exception object, there will more likely be a specific class for a specific kind of exception. This allows the application to invoke a method tailored for handling that exception.

There might be, for example, a class called INVALID-ACCOUNT, whose objects correspond to individual occurrences of the invalid account condition. The application can create and "raise" such an object when the application detects a transaction with an invalid account number. Let's assume the object reference pointing to this object is called AN-INVALID-ACCOUNT. As soon as the application method (assuming it is a method rather than a program) detects the error, it can execute a RAISE AN-INVALID-ACCOUNT statement, or it can pass the exception back to the invoking runtime element with a RAISING AN-INVALID-ACCOUNT phrase on an EXIT METHOD or a GOBACK statement. In the latter case, control is returned to the invoking runtime element, and the exception condition is handled in this invoking element as if the statement that invoked the method had been immediately followed by the statement 'RAISE AN-INVALID-ACCOUNT'.

If the runtime element containing the RAISE or the INVOKE statement contains an "applicable" declarative, for example a declarative for the INVALID-ACCOUNT class, this declarative is then executed. It will typically invoke an appropriate exception-handling method on the exception object – which can be addressed by the EXCEPTION-OBJECT identifier – to issue a warning about the error.

When there is no applicable declarative, the exception may be propagated to a higher level of the invocation hierarchy, if the >>PROPAGATE directive is in effect.

Note that the EXIT METHOD RAISING AN-INVALID-ACCOUNT or the GOBACK RAISING AN-INVALID-ACCOUNT statement returns the exception to the invoking runtime element only if this use of the class INVALID-ACCOUNT has been "announced" by listing that class (or a superclass of it) in the procedure division header of the method containing the EXIT or the GOBACK statement. Otherwise, the predefined exception condition EC-OO-EXCEPTION will be propagated instead, and no detailed analysis of the problem will be possible in the invoking runtime element.

A declarative may alternatively be specified for a specific interface, rather than for a class. In this case, the declarative qualifies for the raised exception object if the object is described with an IMPLEMENTS clause that references that interface.

Similarly, an interface-name rather than a class-name may be specified in the procedure division header of a source element containing an EXIT statement with the RAISING phrase or a GOBACK statement with the RAISING phrase. In this case, the object being raised shall be described with an IMPLEMENTS clause that references the specified interface in order to qualify for propagation.

D.18.10 Sample application

The following is an example of a very simple banking application, consisting of one main program and the Account class. It is not intended to illustrate all the object oriented features of COBOL.

D.18.10.1 Main program

Most object oriented applications have a conventional program to start the processing. BANKMAIN serves this function in this sample bank application.

```
PROGRAM-ID.   BANKMAIN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
```

```
   CLASS Account.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 an-object              USAGE OBJECT REFERENCE Account.
PROCEDURE DIVISION.
go-now.
   INVOKE Account "newAccount" RETURNING an-object.
   INVOKE an-object "displayUI".
   SET an-object to NULL.
   GOBACK.
END PROGRAM BANKMAIN.
```

D.18.10.2 Account class

The source code for the account class is illustrated below. The class has three factory methods:

- newAccount creates a new instance of an account object,
- addAccount adds 1 to the value of number-of-accounts, and
- removeAccount subtracts 1 from the value of number-of-accounts.

The account class also has five instance methods:

- displayUI displays the value of the account balance or performs another function based on a user's request,
- balance retrieves the balance of the account,
- deposit adds an amount to the current balance of the account,
- withdraw subtracts an amount from the current balance of the account,
- initializeAccount moves initial values into the instance data.

```
CLASS-ID.  Account  INHERITS Base.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
   CLASS   Base.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  number-of-accounts           PIC 9(5) VALUE ZERO.
PROCEDURE DIVISION.
METHOD-ID.    newAccount.
DATA DIVISION.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.
01  an-object                    USAGE IS OBJECT REFERENCE ACTIVE-CLASS.
PROCEDURE DIVISION RETURNING an-object.
begin-here.
   INVOKE SELF "new" RETURNING an-object.
   INVOKE an-object "initializeAccount" USING BY CONTENT number-of-accounts.
   EXIT METHOD.
END METHOD newAccount.
METHOD-ID.    addAccount.
PROCEDURE DIVISION.
method-start.
   ADD 1 TO  number-of-accounts.
   EXIT METHOD.
END METHOD addAccount.
METHOD-ID.    removeAccount.
PROCEDURE DIVISION.
main-entry.
   SUBTRACT 1 FROM number-of-accounts.
   EXIT METHOD.
END METHOD removeAccount.
END FACTORY.

OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
```

```
01  account-balance              PIC S9(9)V99.
01  account-number               PIC X(9).
01  the-date                     PIC 9(8).
PROCEDURE DIVISION.
METHOD-ID.    displayUI.
DATA DIVISION.
LOCAL-STORAGE SECTION.
01  in-data.
   03   action-type              PIC X.
   03   in-amount                PIC S9(9)V99.
   03   in-wrk                   PIC X(12).
PROCEDURE DIVISION.
method-start.
   DISPLAY  "Enter D for Deposit, B for Balance or W for Withdrawal"
   ACCEPT in-data
   EVALUATE action-type
      WHEN "D"
         PERFORM get-amount
         INVOKE SELF "deposit" USING in-amount
      WHEN "W"
         PERFORM get-amount
         INVOKE SELF "withdraw" USING in-amount
      WHEN "B"
         INVOKE SELF "balance"
      WHEN OTHER
         DISPLAY  "Enter valid transaction type."
         EXIT METHOD
   END-EVALUATE
   EXIT METHOD
   .
get-amount.
   DISPLAY "Enter amount  9(9).99"
   ACCEPT in-wrk
   COMPUTE in-amount = FUNCTION NUMVAL (in-wrk)
   .
END METHOD  displayUI.

METHOD-ID.    balance.
DATA DIVISION.
LOCAL-STORAGE SECTION.
01   display-balance              PIC $ZZZ,ZZZ,ZZ9.99B—.
PROCEDURE DIVISION.
disp-balance.
   MOVE account-balance to display-balance
   DISPLAY "Your Account Balance is:"  display-balance
   EXIT METHOD.
END METHOD balance.
METHOD-ID.     deposit.
DATA DIVISION.
LINKAGE SECTION.
01   in-deposit                   PIC S9(9)V99.
PROCEDURE DIVISION USING in-deposit.
make-deposit.
   ADD in-deposit TO account-balance
   EXIT METHOD.
END METHOD deposit.

METHOD-ID.   withdraw.
DATA DIVISION.
LINKAGE SECTION.
01   in-withdraw                      PIC S9(9)V99.
PROCEDURE DIVISION USING in-withdraw.
withdraw-start.
   IF account-balance >= in-withdraw
      SUBTRACT in-withdraw  FROM  account-balance
   ELSE
      DISPLAY "Your Balance is Inadequate"
   END-IF
   EXIT METHOD.
END METHOD withdraw.
```

```
METHOD-ID. initializeAccount.
DATA DIVISION.
LINKAGE SECTION.
01  new-account-number  PIC 9(5).
PROCEDURE DIVISION USING new-account-number.
Begin-initialization.
    MOVE ZERO TO account-balance
    MOVE new-account-number TO account-number
    MOVE FUNCTION CURRENT-DATE (1: 8) TO the-date
    EXIT METHOD.
END METHOD initializeAccount.
END OBJECT
END CLASS Account.
```

D.19 Report writer

Report writer is a facility for producing printed outputs that places its emphasis on their organization, format, and content, rather than by specifying the detailed procedures needed to produce them. The report writer language contains a concise data division syntax and, when this is used to define the report structure, relatively little procedural code is needed to produce the printed output, because most of the procedure division programming that would normally be supplied by the programmer is instead provided automatically. The programmer is freed from the task of writing procedures for constructing print lines, moving data, counting lines on a page, numbering pages, producing headings and footings, recognizing the end of logical data subdivisions, accumulating totals, printing tables and testing conditions.

D.19.1 Reports and report files

A report is any printed output with not more than one basic layout for its report headings, report footings, page headings and page footings and one optional hierarchical scheme of control breaks and control groups. More complex printed outputs can be obtained by describing more than one report for the same report file and printing pages from different reports in the order required. For an external file connector referenced by a file-name and for a file shared by multiple file connectors, separate runtime elements may specify different reports for the same file.

Each report is divided vertically into report groups. A report group is a block of print lines, generated in a single operation, that is never split across more than one page.

Report files differ from other output files in that their FD entry has a REPORT clause but no level-01 record descriptions, and their entire content is written through the execution of one of the report writer procedural statements: INITIATE, GENERATE and TERMINATE. Records should not be written to report files in any other manner. The FILE-CONTROL and FD entries and the OPEN and CLOSE statements perform the same function for report files as for other output files. A source unit may contain more than one report file. In the following example, two reports are defined:

```
FD  SUMMARY-PRINT-FILE
        REPORTS ARE DETAILED-LISTING, SUMMARY-PRINT.
```

D.19.2 RD entry

Each RD entry assigns a name to one report and describes its general characteristics. Its optional clauses are as follows:

D.19.2.1 PAGE

The PAGE clause defines the layout of a physical report page. The COLUMNS phrase defines the width of the page. The first printable line on the page is considered to be line 1, and all other phrases are based in relationship to this. The LINES phrase defines the position of the last print line on the page.

The HEADING phrase defines where the first line of the page or report heading will be printed. If the HEADING phrase is absent, printing begins on line 1. The first line of a control group will print on either the line specified in the FIRST DETAIL phrase or one line after the last line of the last report or page heading printed, whichever is greater.

The LAST CONTROL HEADING and the LAST DETAIL phrases determine when a page advance takes place. If the next item to be printed is the control heading, and the number of lines necessary to print the control heading added to the current line count for the page is greater than specified in the LAST CONTROL HEADING phrase a page advance will occur. If the next item to be printed is a detail item and the number of lines necessary to print the detail added to the current line count is greater than specified in the LAST DETAIL phrase, a page advance will occur.

The FOOTING phrase has two different functions:

1) If the next item to be printed is a control footing, and the number of lines necessary to print the footing added to the current line count is greater than specified in the FOOTING phrase, a page advance will occur.

2) When a page advance occurs, the line count is advanced to either the position specified in the FOOTING phrase or to the next line, whichever is greater, and the applicable page and report footings are printed.

	PAGE clause phrases	Report content
Physical Top of Form		the first line that can be printed
		<blank>
Logical Top of Form	**Heading**	the first line of the report or page heading <(first page only) report header lines> <page heading lines>
	First Detail	the first line of a body group <Control Heading lines> <Detail lines> <Control Footing lines> <Control Heading lines> <Detail lines> <Control Footing lines>
		<Control Heading lines> <Detail lines> <Control Footing lines>
	Last Control Heading	the last line on which a control heading may print <Detail lines>
	Last Detail	the last line on which Detail lines will print <Control Footing lines>
	Footing	last line of Control footing or first line of a report or page footing <page footing lines. <(last page only) report footing lines>
Logical Bottom of Form	**Page Limit**	the last line on a logical page
Physical Bottom of Form		the last line that can be printed

Figure C.6 — Example of page layout

D.19.2.2 CONTROL

The CONTROL clause defines a hierarchy of data-names whose values will be used to sense for control breaks. FINAL may be specified as the highest-level control. For each data-name a control heading and a control footing group may be specified, which will be printed automatically on each occasion that a control break at that level, or above, is detected.

D.19.2.3 CODE

The CODE clause specifies information that is not to be printed that can be used to separate multiple reports that are written to the same file or may be needed for correct functioning of the printing device. For example, suppose

you write two reports to a file and specify CODE "A" for one and CODE "B" for another. If your system allows reading of the report file you can separate both reports by reading the file, removing the first character, and sending A lines to one place and B lines to another.

D.19.2.4 Example

The following example shows one instance of each of these optional clauses in the RD entry:

```
RD  SUMMARY-PRINT
    PAGE LIMITS ARE 60 LINES
       HEADING 2
       FIRST DETAIL 5
       LAST DETAIL 56
       FOOTING 58
    CONTROLS ARE FINAL, WS-YEAR, WS-MONTH
    CODE IS REFERENCE-NO.
```

D.19.3 Basic report group description

D.19.3.1 TYPE

The RD entry is followed by any number of report group descriptions, each beginning with a level-01 entry containing a TYPE clause. The TYPE clause defines how the report group will be used in the report. The order in which the report group descriptions are coded is immaterial.

D.19.3.2 LINE and NEXT GROUP

The lines of the report group correspond to LINE clauses in the report group description. LINE clauses are not nested and are written at a subordinate level (although a single-line report group may be described using a LINE clause in the level-01 entry).

The operands of the LINE clauses specify the vertical positioning of the lines on the page. Lines can be absolute or relative. Absolute lines are defined by an integer with an optional NEXT PAGE phrase. Relative lines are indicated by the word PLUS or + before the integer. The absolute form gives the line number with respect to the top of the printed page, whereas the principal effect of the relative form is to move a vertical distance with respect to the previous line. Additional spacing after the report group has been printed may be indicated by using the NEXT GROUP clause, which has similar absolute and relative forms, and an additional NEXT PAGE phrase.

D.19.3.3 COLUMN

COLUMN clauses are used to specify the horizontal location of elementary items. COLUMN may be shortened to COL. Items whose description entries contain a COLUMN clause are referred to as printable items. Provided they are not absent as a result of a PRESENT WHEN clause, they are always printed when the report group containing them is printed. The COLUMN clause can be absolute, indicated by an integer, or relative, indicated by the word PLUS or +, before the integer. The absolute form gives the column number with respect to the left side of the printed page, whereas the relative form moves the item the specified horizontal distance from the last character of the previous printable item. Spaces are supplied in all unused columns. Elementary items whose description entries do not contain a COLUMN clause are referred to as unprintable items. They are not printed but may be referred to in SUM clauses for the purpose of totaling.

D.19.3.4 SOURCE, VALUE, and PICTURE

The establishing of the contents of elementary report items is directed not by procedure division statements but by the report section clauses SOURCE, SUM and VALUE. SUM is described in more detail below under "Totaling".

SOURCE specifies the sending operand of an implicit MOVE or COMPUTE statement and the report entry that contains it defines the receiving printable item.

VALUE assigns a fixed literal to the printable item. The content of such an item may be modified only by means of a PRESENT WHEN clause.

A PICTURE clause is used, in the same way as any other receiving operand, to indicate the editing requirements for printable items. The PICTURE clause is optional when the elementary item has a clause of the form VALUE IS "literal".

D.19.3.5 Example:

The following example illustrates the basic report group description clauses:

```
01  TYPE PAGE HEADING.
    05  LINE 2.
        07  COL CENTER 40   VALUE "NAME OF MY COMPANY".
        07  COL 74          VALUE "Page".
        07  COL RIGHT 80    PIC ZZ9 SOURCE PAGE-COUNTER.
    05  LINE + 2.
       07  COLS 5 30 40     VALUES "Description" "Date" "Value".
01  ORDER-LINE  TYPE DETAIL.
    05  LINE + 2.
        07  COL 5               PIC X(20)  SOURCE ORDER-DESCRIPTION.
        07  COL 30              PIC 9(6)  SOURCE ORDER-DATE.
        07  COL 40              PIC ZZZ,ZZ9.99—
                                SOURCE (ORDER-VALUE — SPECIAL-DISCOUNT) / 100.
```

D.19.4 Modifying the report group layout

D.19.4.1 PRESENT WHEN

The PRESENT WHEN clause specifies a condition that, if false, suppresses, for that instance of the report group, the processing of the report section entry, whether it be that of a report group, line, elementary report item, or some intermediate-level item. If the item suppressed is not elementary, all its subordinate items are also suppressed.

The concept of suppressing an item does not necessarily imply that spaces are printed. Rather, the effect of suppressing an item is that the report behaves at that instant as though the item's data description were omitted entirely from the report description. The following effects are a consequence of this principle:

1) Absent lines are not printed, do not alter the LINE-COUNTER, and do not contribute to the vertical size of the corresponding report group when the page fit test is performed.

2) Absent printable items do not alter the horizontal counter in the current line. Consequently, a printable item whose entry has a PRESENT WHEN clause affects the column position of any relative printable items that immediately follow it in the report line.

3) Within a report group, lines may apparently overlap or overflow the permitted region of the page, provided that their LINE clauses are subject to PRESENT WHEN clauses specifying mutually exclusive conditions, so that, after absent lines have been eliminated, such overlap or overflow does not occur.

4) Within a given line, printable items may apparently overlap or, if relative, apparently exceed the page width, provided that their COLUMN clauses are subject to PRESENT WHEN clauses specifying mutually exclusive conditions, so that, after absent items have been eliminated, such overlap or overflow does not occur.

It is not necessary to specify an alternate blank line or blank printable item to define the case where a line or printable item is absent, because unoccupied lines on the page and unoccupied columns in the line are always blank.

The PRESENT WHEN clause is illustrated in the example below. If FIRST-NAME contains "UNKNOWN", spaces are printed instead of FIRST-NAME up to column 25 because SURNAME is printed in absolute column 25. However, if the MEMBER-FLAG does not contain 1, both MEMBER-NO and MEMBER-TYPE vanish and CITY is printed in column 47, because its COLUMN clause is relative.

```
05  LINE + 2.
    07  COL 1      PIC X(20)    SOURCE FIRST-NAME
                                PRESENT WHEN FIRST-NAME NOT = "UNKNOWN".
    07  COL 25     PIC X(20)    SOURCE SURNAME.
     07  PRESENT WHEN MEMBER-FLAG = 1.
         09  COL 51   PIC X(6)    SOURCE MEMBER-NO.
         09  COL 61   PIC X(10)   SOURCE MEMBER-TYPE.
     07  COL + 3    PIC X(10)     SOURCE CITY.
```

In the next example, the second line will be suppressed entirely if BANK-FLAG is not 1. If these lines are part of a body group, the suppressed line is not included in the body group's vertical size when any page fit test is applied. By contrast, if the LINE clauses were absolute, a blank line would result in the place of the suppressed line.

```
05  LINE + 2.
     07  COL 1      PIC 9999    SOURCE PAYMENT.   etc.
05  LINE + 1     PRESENT WHEN BANK-FLAG = 1.
     07  COL 1      PIC X(8)    SOURCE BANK-ACCOUNT.   etc.
```

D.19.4.2 GROUP INDICATE

The GROUP INDICATE clause is designed to enable a data item (typically a control data item) to be printed once only at the start of a series of details following a control break or page break. It behaves like a special case of PRESENT WHEN.

D.19.5 Repetition

D.19.5.1 OCCURS

The OCCURS clause enables any part of a report group defined by an entry below level 01, whether it be a line, elementary item, or some item described at an intermediate level, to be repeated a given number or variable number of times. OCCURS clauses may be nested to print multi-dimensional tables. The STEP phrase gives the horizontal or vertical spacing between items and is necessary if the item has an absolute (horizontal or vertical) position.

```
03  LINE 4       OCCURS 4  STEP 1.
 *> The VALUE clause applies to each repetition
    05  COL 1    OCCURS 5  STEP 10    VALUE "PAY".
03  LINE + 1     OCCURS 2.
 *> The SOURCE clause also applies to each repetition
    05  COL + 6  OCCURS 3  PIC X(4)  SOURCE IS WS-CODE.
```

D.19.5.2 Multiple form of LINE, COLUMN, SOURCE, VALUE

Instead of coding an OCCURS clause, the programmer may code a multiple COLUMN or LINE clause, an indispensable facility when the distance between items is irregular.

```
03  LINES ARE 4, 5, 7, + 1.
    05  COLUMNS ARE 1, 11, 22, +5 ...
```

In order to place different values or different source data items in successive entries, a multiple SOURCE or VALUE clause may be used. This is valid whether an OCCURS clause or a multiple LINE or COLUMN clause is used to establish the repetition:

```
03  LINES 2, 3.
    05  COLS  ARE  13, 35, 56
        VALUES ARE "THIS"  "NEXT"  "SOME" "YEAR"  "YEAR"  "DAY".
03  LINE + 1.
    05  COL 12  OCCURS 3   STEP 22  PIC ZZZ9
        SOURCES ARE WS-PAY WS-BONUS WS-REFUND.
```

D.19.5.3 VARYING

The VARYING clause may be used with an OCCURS clause or a multiple LINE or COLUMN clause to establish a repetition counter. This counter may then be used as a subscript or for any other purpose.

```
       03  LINES 2, 3, 4     VARYING RS-LINE-NO FROM 6 BY 6.
           05  COL 1  STEP 10  OCCURS 6
                               VARYING RS-ENTRY-NO FROM RS-LINE-NO BY –1
                               PIC ZZZ9
                               SOURCE IS WS-PAY-TABLE (RS-ENTRY-NO)
                               PRESENT WHEN WS-PAY-TABLE (RS-ENTRY-NO) >= 0.
```

D.19.6 Totaling

The SUM clause may be placed in a printable item's description instead of SOURCE or VALUE. It causes the indicated numeric data items to be accumulated in an internal sum counter. When the item is printed, the accumulated value is moved into the printable item and (except when a RESET phrase is specified) reset to zero once the processing of any remaining entries in the report group description is complete.

A sum counter is an internal location, defined automatically. The number of its integral and fractional digits are implied by the PICTURE clause used with the SUM clause. A sum counter is always signed. A program may have any number of sum counters.

The operands of the SUM clause may either be data items defined in the report section or data items defined in some other part of the data division. If a SUM operand is defined in the report section, it is added into the sum counter whenever the report group in which it is defined is printed. If a SUM operand is not defined in the report section, it is added into the sum counter whenever a GENERATE is executed for that report. For the latter case, an UPON phrase is provided to control which GENERATE statements cause adding and which do not.

If the SUM operand is an array (a repeating data item) then the SUM entry may be an array of the same size (in which case an array of totals is formed) or, if it is a single data item or a table with fewer dimensions than the SUM operand, it will add together all the repeating entries in one or more (vertical or horizontal) directions.

The RESET phrase may be used to delay the resetting of sum counters, thus providing cumulative totals.

An entry with a SUM clause may be unprintable (without a COLUMN clause), so that a total may be formed but not immediately printed. The current value of a sum counter may be obtained by referencing the data name of the entry.

In the following example, the SUM clause is used to produce subtotals at two levels.

```
RD  PAY-REPORT
    CONTROLS ARE WS-YEAR WS-MONTH.

01  EMPLOYEE-PAYMENT          TYPE DETAIL.
    03  LINE + 1.
        05  RS-PAY  COL 4  STEP 10  OCCURS 4 TIMES
 *> This repeating entry could use a table as its SOURCE instead,
 *> VARYING a data name as a subscript as in the previous example under VARYING.
                               PIC Z(3)9    SOURCES WS-PAY WS-BONUS WS-EXTRA
                                                    WS-REFUND.
 *> This entry accumulates the 4 items in a single total.
        05  RS-EMPLOYEE-TOTAL-PAY   COL 51  PIC Z(6)9
            SUM OF RS-PAY.

01  TYPE CF FOR WS-MONTH.
    03  LINE + 1.
 >* This entry gives 4 totals for the 4 items added vertically
        05  COL 1  STEP 10  OCCURS 4  PIC Z(6)9
            SUM OF RS-PAY.
        05  RS-MONTH-TOTAL-PAY      COL 51    PIC Z(6)9
 *> SUM OF RS-PAY would give the same result here:
            SUM OF RS-EMPLOYEE-TOTAL-PAY.
```

```
 *> This entry will print a cumulative total
        05  RS-MONTH-CUMULATIVE-PAY COL 1     PIC Z(7)9
            SUM OF RS-EMPLOYEE-TOTAL-PAY   RESET ON WS-YEAR.

01  TYPE CF FOR WS-YEAR.
    03  LINE + 1.
 *> The following item is unprintable:
        05  RS-YEAR-TOTAL-PAY                 PIC Z(7)9
 *> SUM OF RS-EMPLOYEE-TOTAL-PAY would give the same result here:
            SUM OF RS-MONTH-TOTAL-PAY.
 *> This entry uses the yearly total in a source, scaling it down
        05  COL 1   PIC Z(5)9.99
            SOURCE IS RS-YEAR-TOTAL-PAY / 100 ROUNDED.
```

D.19.7 Procedure division statements

Although the description of the report groups and most of the physical and logical organization of a report is defined using data division elements, these descriptions do not have any effect on the processing of the report except during the execution of one of the procedure division statements INITIATE, GENERATE and TERMINATE. The processing of a report consists of the following steps:

1) The file connector with which the report is associated shall be in an open mode.

2) The INITIATE statement is executed to initiate processing of the report.

3) Once the report is initiated, the GENERATE statement is used to generate the various parts of the report. Each GENERATE statement causes one detail group to be printed (except when the GENERATE report-name form of the statement is used), possibly preceded by other types of report groups resulting from a control break or page fit test.

4) When the report is to be completed, the TERMINATE statement is executed to finish it.

5) The file connector may then be closed or left open for more reports.

A typical sequence of execution might be as follows:

```
OPEN OUTPUT report-file
INITIATE report-name
obtain input data record(s)
PERFORM UNTIL <end of input data>
   GENERATE detail-A
   GENERATE detail-B, etc.
   obtain input data record(s)
END-PERFORM
TERMINATE report-name
CLOSE report-file
```

D.19.8 Report counters

The PAGE-COUNTER and LINE-COUNTER identifiers refer to locations that are defined automatically for each report description entry. They contain, respectively, the number of the current page and the number of the current line within the page.

D.20 Structured constant

A "structured constant" is a record described with the CONSTANT RECORD clause at level 1.

The contents of a structured constant are fixed and cannot be changed once the program has been compiled, either during program activation or during the lifetime of execution of the program. A structured constant is a read-only item. Use of a structured constant or any data item within it as a receiving data item, either by direct access or indirect means, is prohibited.

A structured constant and its subordinate items are data items and can be used anywhere a sending data item can be used. This differs from a constant entry, which has no structure, and can be used only where a literal of the corresponding type can be used.

The following example illustrates the format and content of a structured constant:

```
1  A-DATA-ITEM CONSTANT RECORD.
   2  FIELD-1 BINARY PICTURE S9(31).
   2  FIELD-2 DISPLAY PICTURE X(50).
   2  ARRAY-INIT PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
   2  DISPLAY-CHARS REDEFINES ARRAY-INIT.
      3  DISPLAY-CHAR PICTURE X OCCURS 26 TIMES.
   2  FILLER PIC X(3) VALUE ALL "Q".
   2  FILLER PIC X(4).
```

The content of A-DATA-ITEM does not change during the lifetime of the program in which it is declared. The content of A-DATA-ITEM and its subordinate fields are those specified for the applicable VALUE clauses, if any, and for the data items that do not have VALUE clauses, the default appropriate to the USAGE of the field, as described in the INITIALIZE statement.

In this example, FIELD-1 will contain zeroes; FIELD-2 will contain spaces; ARRAY-INIT will contain "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; the first FILLER data item will contain "QQQ"; and the second FILLER item will contain four space characters.

D.21 Validate facility

The validate facility provides a major procedural statement VALIDATE and several associated data division clauses that enable data to be checked for various errors and inconsistencies in a high-level and comprehensive manner.

The VALIDATE statement can be used to perform validation on any data item defined in the file, working-storage, local-storage, or linkage section. The item is checked for conformance to its data description and messages or flags of the programmer's choice may be issued or set in response. Data items can also be stored automatically in target locations.

The detailed processing undertaken by the VALIDATE statement is established entirely by the description of the referenced data item rather than by code in the procedure division. As well as general data division clauses, such as PICTURE, additional clauses may be included that control the action of the VALIDATE statement, but have no effect otherwise.

The data item may be a group or elementary item of any length and complexity. The operation of the VALIDATE statement is divided into several stages. Each stage is completed for the entire data item, including all its subordinate items, before the next stage begins. If an elementary data item fails one stage of validation it cannot be rejected at any further stage. If a check fails, there is no interruption to processing: instead, errors are indicated by the storing of messages or indicators defined by the programmer. The stages are as follows:

— Format validation
— Input distribution
— Content validation
— Relation validation
— Error indication

D.21.1 Format validation

This stage uses the PICTURE clauses, possibly modified in their meaning by the SIGN and USAGE clauses, to check that each data item has the expected data format. An elementary item is assigned a default value if it fails this check or if its usage is display or national and its value is all spaces. The default value is used in subsequent references to the item. A group item may also have a DEFAULT clause.

The clauses relevant to this stage are: PICTURE, DEFAULT, USAGE, SIGN, and VARYING.

D.21.2 Input distribution

This stage activates any DESTINATION clauses that are defined for the data item to store items in their target locations where indicated. By virtue of the default values, target locations always receive valid data, unless DEFAULT NONE is specified when they are unchanged.

The clauses relevant to this stage are: DESTINATION and VARYING.

D.21.3 Content validation

This stage checks that each data item lies in the expected set of permissible values. 88-level entries may be used to check the values of the data item as a whole and the CLASS clause to check each of its characters. The range of values specified in 88-level entries can depend on specified conditions.

The clauses relevant to this stage are: CLASS, VALUE and VARYING.

D.21.4 Relation validation

This stage checks that data items have appropriate values in relation to any other data items. Other terms for this stage are "inter-field checks" or "cross-field validation". The contents of a data item may be considered to be invalid, depending upon the truth value of a specified condition. For example, the clause 'ITEM-A INVALID WHEN

ITEM-B > 35' means that the contents of ITEM-A are always considered to be invalid when the value of ITEM-B is greater than 35.

The clauses relevant to this stage are: INVALID and VARYING.

D.21.5 Error indication

This final stage operates on any VALIDATE-STATUS clauses that are defined and uses them to set up messages or indicators, specified by the programmer, to identify which items, if any, have been rejected. It is also possible to distinguish between the three reasons for rejection. The VALIDATE-STATUS clauses are not placed within the data item being validated but elsewhere in the data division.

The clauses relevant to this stage are: VALIDATE-STATUS and VARYING.

D.21.6 Validation of more complex formats

The data description can contain subordinate entries that have an OCCURS clause. Each repetition of the data item can then be checked independently, can have an independent DESTINATION, and can be assigned an independent error message or indicator. If the data description has alternate formats, these can be described using the REDEFINES clause and the appropriate description can be selected using the PRESENT WHEN clause.

The clauses relevant to these formats are: PRESENT WHEN, REDEFINES, OCCURS, and VARYING.

D.21.7 Examples of validation

The comments placed in these examples explain the use and effects of the various clauses.

D.21.7.1 Example of validation of USAGE DISPLAY items

```
 01  INPUT-RECORD.
*>PIC 99 checks that IN-TYPE is 2 characters numeric;
     03  IN-TYPE         PIC 99
*>if IN-TYPE fails the PICTURE check, it is assumed to be 1;
*>without a DEFAULT clause, the assumed value would here be 0.
                           DEFAULT 1.
*>PRESENT WHEN states the condition for this format to be used.
     03  IN-REC-FORMAT-1 PRESENT WHEN IN-TYPE = 0 OR 1 OR 2.
*>PICTURE A(20) checks for 20 alphabetic (or space) characters.
         05  IN-NAME       PIC A(20)
*>PRESENT WHEN defines when the validation clauses for this data item apply:
                             PRESENT WHEN IN-TYPE = 0 OR 1
*>CLASS checks each character for a class defined in SPECIAL-NAMES.
                             CLASS IS ALPHA-UPPER
*>DESTINATION moves this item (or spaces if not alpha) to OUT-NAME.
                             DESTINATION OUT-NAME.
*>PRESENT WHEN checks whether the item is "blank" under this condition
         05  FILLER REDEFINES IN-NAME
                             PRESENT WHEN IN-TYPE = 2
                             DESTINATION OUT-NAME.
             88              VALUE SPACES IS VALID.
*>The values of IN-WEEK are checked to be in non-descending order.
         05  IN-WEEK       PIC 99  OCCURS 5
             VARYING IN-WEEK-NO FROM 1, IN-NEXT-WEEK-NO FROM 2
             INVALID WHEN IN-WEEK-NO < 5
             AND IN-WEEK (IN-WEEK-NO) > IN-WEEK (IN-NEXT-WEEK-NO)
*>OUT-WEEK (1) to (5) will hold the values of IN-WEEK (1) to (5),
*>or zero for any one that failed the format (PICTURE) test.
             DESTINATION OUT-WEEK (IN-WEEK-NO).
*>The 88-level INVALID entries check for invalid ranges of values.
            88  VALUES 0, 53 THRU 99 ARE INVALID.
*>REDEFINES and another PRESENT WHEN define an alternate format.
     03  IN-REC-FORMAT-2 REDEFINES IN-REC-FORMAT-1
                                   PRESENT WHEN IN-TYPE > 2.
*>IN-PAY has insertion characters that must be present on input.
```



```
         05  IN-PAY               PIC ZZ,ZZZ.ZZ.
*>The 88-level VALID entries check for valid ranges of values;
*>the condition-name, if present, may be used in the usual way.
*>The following assume that DECIMAL POINT IS COMMA is not specified.
             88  IN-PAY-OK  VALUES "10,000.00" THRU "20,000.00" ARE VALID.
*>88-level entries may also have a condition attached.
             88                   VALUES "20,000.01" THRU "30,000.00" ARE VALID
                                  WHEN IN-TYPE = 8.
*>exceptional cases can be specified using PRESENT WHEN
         05  IN-CODE              PIC AX(3)9(4)
                                  PRESENT WHEN IN-CODE NOT = "UNKNOWN".
         05  FILLER          PIC X(13).
*>
*>*************************************************************
*>Description of target record
*>*************************************************************
*>This is set up by the optional DESTINATION clauses defined
*>in the input record;
*>if a format error is found, a default value is stored instead.
 01  TARGET-AREA.
     05  OUT-NAME       PIC X(20).
     05  OUT-WEEK       PIC 99 COMP   OCCURS 5.
*>
*>*************************************************************
*> Description of error messages
*>*************************************************************
*>Error messages or flags are set up or cleared automatically
*>when the VALIDATE statement is executed; the programmer chooses
*>where they go and what messages or values they contain;
*>they need not be contiguous as they are in this example.
01  VALIDATE-MESSAGES.
    03  PIC X(40)  VALIDATE-STATUS "Unknown Record Type  -  1 assumed"
                     WHEN ERROR FOR IN-TYPE
*> more than one VALIDATE-STATUS clause may be defined in one entry;
*> a NO ERROR  phrase produces a message when the item is valid.
                   VALIDATE-STATUS "Record type Accepted"
                     WHEN NO ERROR FOR IN-TYPE.
*> The VALIDATE-STATUS clause can pinpoint the stage of the failed check.
    03  PIC X(40)  VALIDATE-STATUS "Name not alphabetic"
                     WHEN ERROR ON FORMAT FOR IN-NAME
                   VALIDATE-STATUS "Lower-case not allowed in name"
                     WHEN ERROR ON CONTENT FOR IN-NAME
                   VALIDATE-STATUS "Name not allowed in this case"
                     WHEN ERROR ON RELATION FOR IN-NAME.
*> If no message is stored, spaces will be stored in these cases.
*> Errors may also be indicated by flags;
*> they may also refer to a table of input items.
    03  W-ERROR-FLAG   PIC 9   COMP  OCCURS 5
                   VALIDATE-STATUS 1 WHEN ERROR FOR IN-WEEK.
*>An EC-VALIDATE (nonfatal) exception is also set if the
*>VALIDATE statement detects an invalid condition.
*>
*>*************************************************************
*>Execution of the VALIDATE statement
*>*************************************************************
 PROCEDURE DIVISION.
 ...
*>A single VALIDATE statement performs all the actions implied
*>in the above data descriptions.
     VALIDATE INPUT-RECORD
*>After this statement has been executed:
*>(1) the input record is unchanged;
*>(2) input items are moved automatically to the target area;
*>(3) error messages are set up wherever specified in the program.
*>
```

D.21.7.2 Example of validation of non-display items

```
         01  MIXED-GROUP TYPEDEF STRONG.
             05  FLD-1             PIC S9(4)  USAGE COMP.
             05  FLD-2             PIC S9(7)  USAGE PACKED-DECIMAL.
             05  FLD-3             PIC 1(8)   USAGE BIT ALIGNED.
             05  PTR-1             USAGE INDEX.
             05  PTR-2             USAGE OBJECT REFERENCE.
             05  TXT-1             PIC N(12)  USAGE NATIONAL.
       01 MY-MIXED-GROUP TYPE MIXED-GROUP.

       PROCEDURE DIVISION.
       ...
      *>A declarative section could be used instead of VALIDATE-STATUS clauses
      *>especially if errors are not expected.
       >> TURN EC-VALIDATE CHECKING ON
           VALIDATE MY-MIXED-GROUP
```

D.22 Conditional expressions

The following figures illustrate how conditional expressions are evaluated.

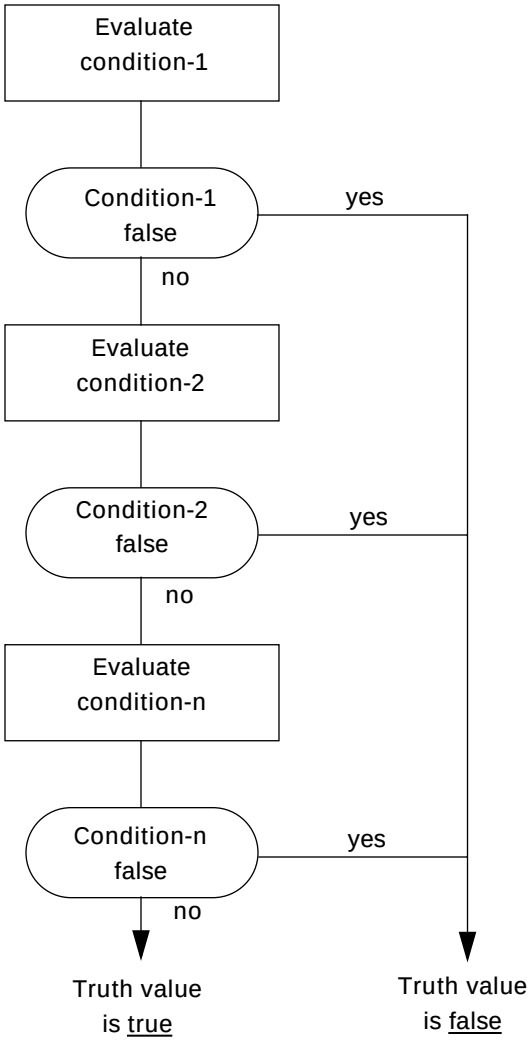


Figure C.7 — Evaluation of the condition-1 AND condition-2 AND ... condition-n

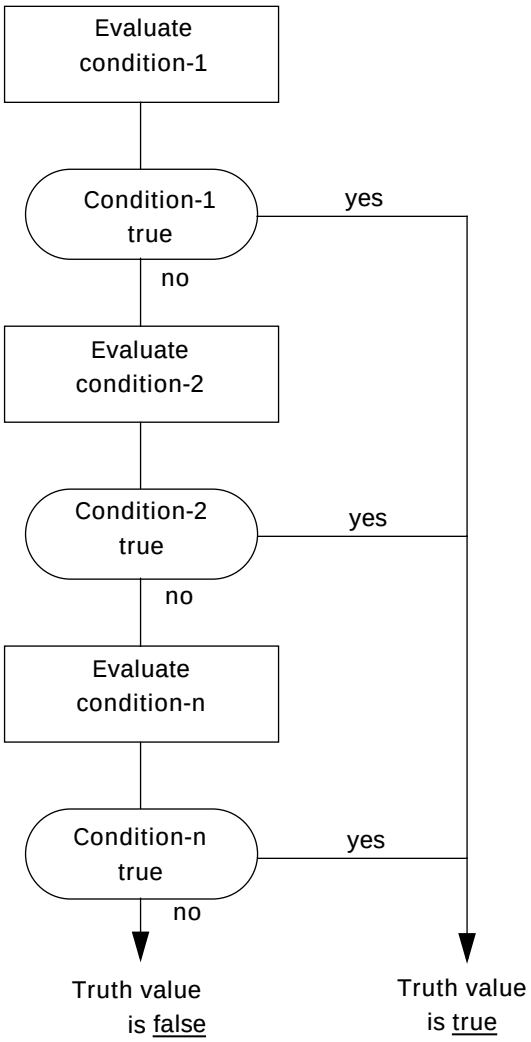


Figure C.8 — Evaluation of the condition-1 OR condition-2 OR ... condition-n

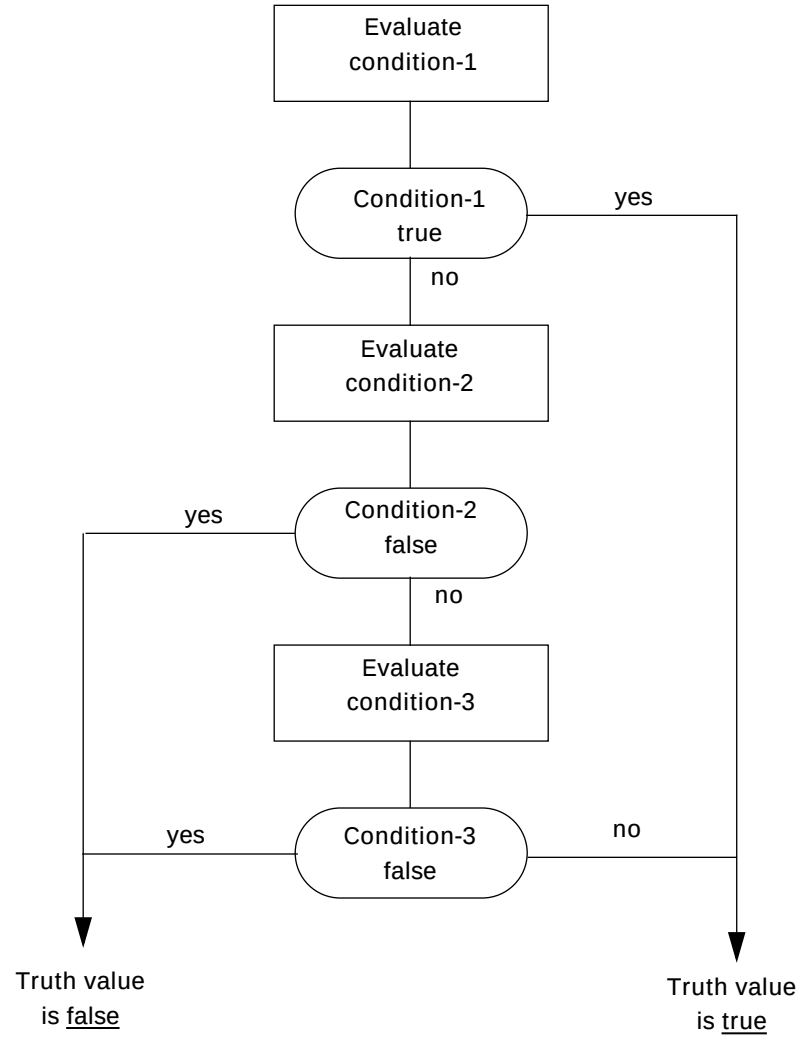


Figure C.9 — Evaluation of condition-1 OR condition-2 AND condition-3

.

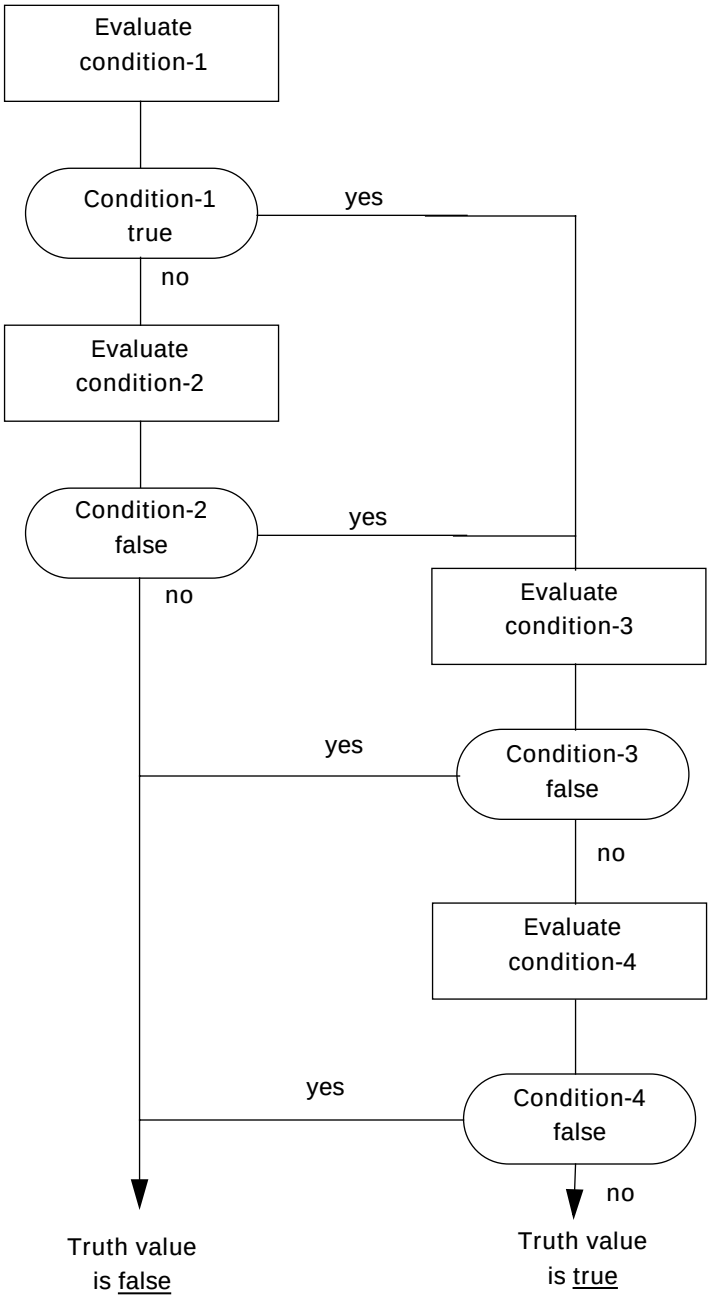


Figure C.10 — Evaluation of (condition-1 OR NOT condition-2) AND condition-3 AND condition-4

D.23 Examples of the execution of the INSPECT statement

In each of the following examples of the INSPECT statement, COUNT-n is assumed to be zero immediately prior to execution of the statement. The results shown for each example, except the last, are the result of executing the two successive INSPECT statements shown above them.

Example 1:

```
INSPECT ITEM TALLYING
   COUNT-0 FOR ALL "AB", ALL "D"
   COUNT-1 FOR ALL "BC"
   COUNT-2 FOR LEADING "EF"
   COUNT-3 FOR LEADING "B"
   COUNT-4 FOR CHARACTERS;
INSPECT ITEM REPLACING
   ALL "AB" BY "XY", "D" BY "X"
   ALL "BC" BY "VW"
   LEADING "EF" BY "TU"
   LEADING "B" BY "S"
   FIRST "G" BY "R"
   FIRST "G" BY "P"
   CHARACTERS BY "Z"
```

Initial Value of ITEM	COUNT-0	COUNT-1	COUNT-2	COUNT-3	COUNT-4	Final Value of Item
EFABDBCGABEFGG	3	1	1	0	5	TUXYXVWRXYZZPZ
BABABC	2	0	0	1	1	SXYXYZ
BBBC	0	1	0	2	0	SSVW

Example 2:

```
INSPECT ITEM TALLYING
   COUNT-0 FOR CHARACTERS
   COUNT-1 FOR ALL "A";
INSPECT ITEM REPLACING
   CHARACTERS BY "Z"
   ALL "A" BY "X"
```

Initial Value of ITEM	COUNT-0	COUNT-1	Final Value of ITEM
BBB	3	0	ZZZ
ABA	3	0	ZZZ

Example 3:

```
INSPECT ITEM TALLYING
   COUNT-0 FOR ALL "AB" BEFORE "BC"
   COUNT-1 FOR LEADING "B" AFTER "D"
   COUNT-2 FOR CHARACTERS AFTER "A" BEFORE "C";

INSPECT ITEM REPLACING
   ALL "AB" BY "XY" BEFORE "BC"
   LEADING "B" BY "W" AFTER "D"
   FIRST "E" BY "V" AFTER "D"
   CHARACTERS BY "Z" AFTER "A" BEFORE "C"
```

Initial Value of ITEM	COUNT-0	COUNT-1	COUNT-2	Final Value of ITEM
BBEABDABABBCABEE	3	0	2	BBEXYZXYXYZCABVE
ADDDDC	0	0	4	AZZZZC
ADDDDA	0	0	5	AZZZZZ
CDDDDC	0	0	0	CDDDDC
BDBBBDB	0	3	0	BDWWWDB

Example 4:

```
INSPECT ITEM TALLYING
   COUNT-0 FOR ALL "AB" AFTER "BA" BEFORE "BC";
INSPECT ITEM REPLACING
   ALL "AB" BY "XY" AFTER "BA" BEFORE "BC"
```

Initial Value of ITEM	COUNT-0	Final Value of ITEM
ABABABABC	1	ABABXYABC

Example 5:

```
INSPECT ITEM CONVERTING
   "ABCD" TO "XYZX" AFTER QUOTE BEFORE "#".
```

The above INSPECT is equivalent to the following INSPECT:

```
INSPECT ITEM REPLACING
   ALL "A" BY "X" AFTER QUOTE BEFORE "#"
   ALL "B" BY "Y" AFTER QUOTE BEFORE "#"
   ALL "C" BY "Z" AFTER QUOTE BEFORE "#"
   ALL "D" BY "X" AFTER QUOTE BEFORE "#".
```

Initial Value of ITEM	Final Value of ITEM
AC"AEBDFBCD#AB"D	AC"XEYXFYZX#AB"D

Example 6:

```
INSPECT ITEM CONVERTING "ABCDEFGHIJKLMNOPQRSTUVWXYZ"-
   "abcdefghijklmnopqrstuvwxyz"
   TO ALL "?"
```

Initial Value of ITEM	Final Value of ITEM
415-245-1212	415-245-1212
415-CH5-1212	415-??5-1212
20%Numeric	20%???????

D.24 Examples of the execution of the PERFORM statement

Representations of the actions of several types of PERFORM statements with varying-phrase specified are given in figures C.11, The VARYING phrase of a PERFORM statement with the TEST BEFORE phrase having one condition, C.12, The VARYING phrase of a PERFORM statement with the TEST BEFORE phrase having two conditions, C.13, The VARYING phrase of a PERFORM statement with the TEST AFTER phrase having one condition, and C.14, The VARYING phrase of a PERFORM statement with the TEST AFTER phrase having two conditions. These are not intended to dictate implementation.

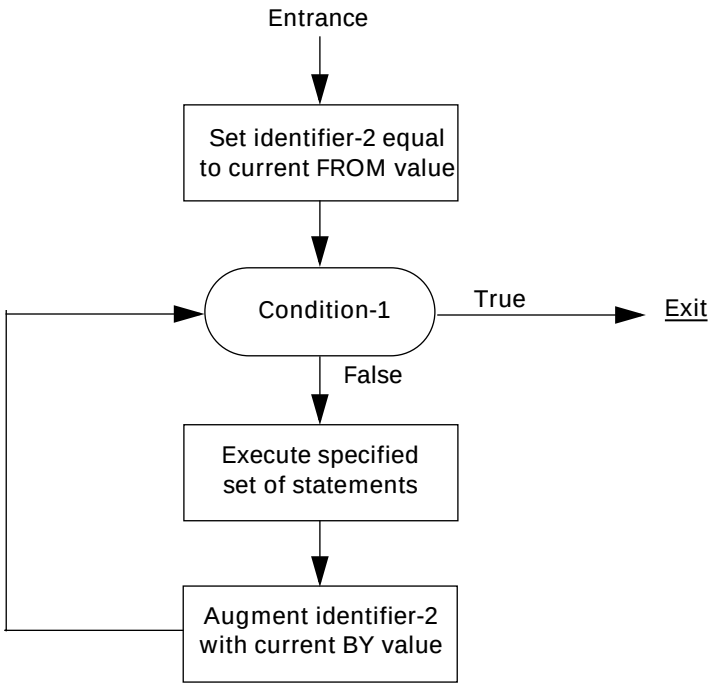


Figure C.11 — The VARYING phrase of a PERFORM statement with the TEST BEFORE phrase having one condition

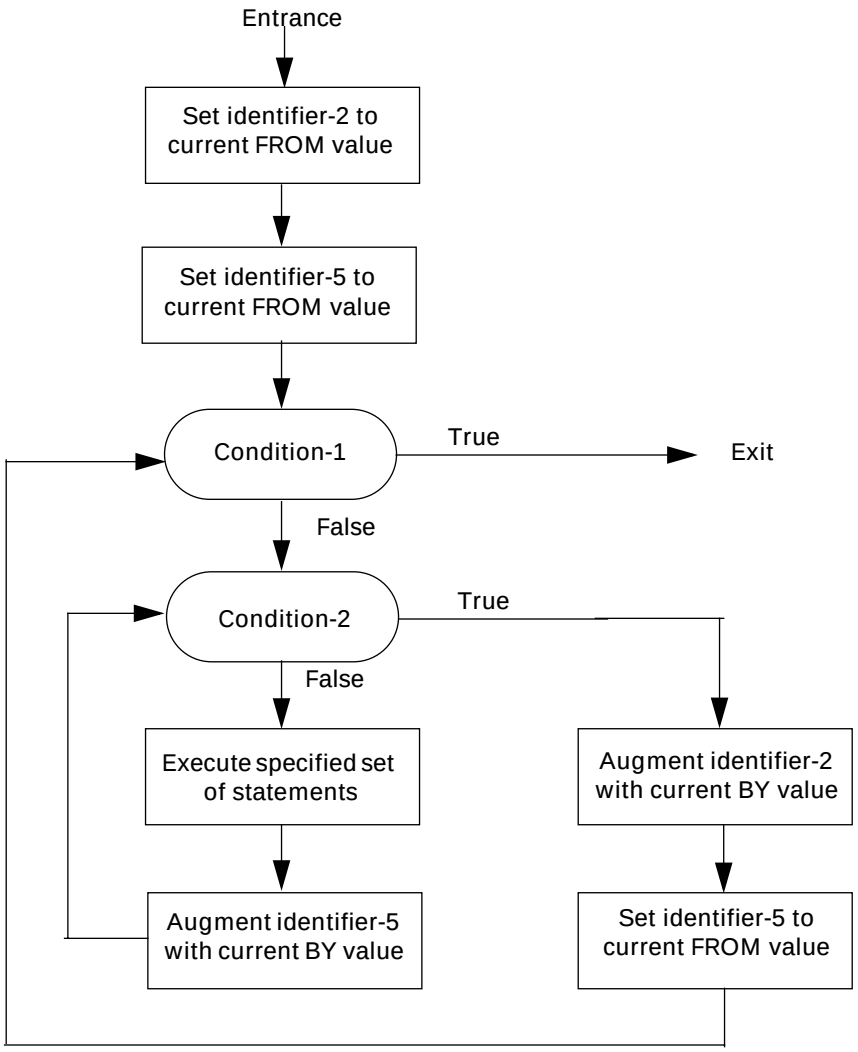


Figure C.12 — The VARYING phrase of a PERFORM statement with the TEST BEFORE phrase having two conditions

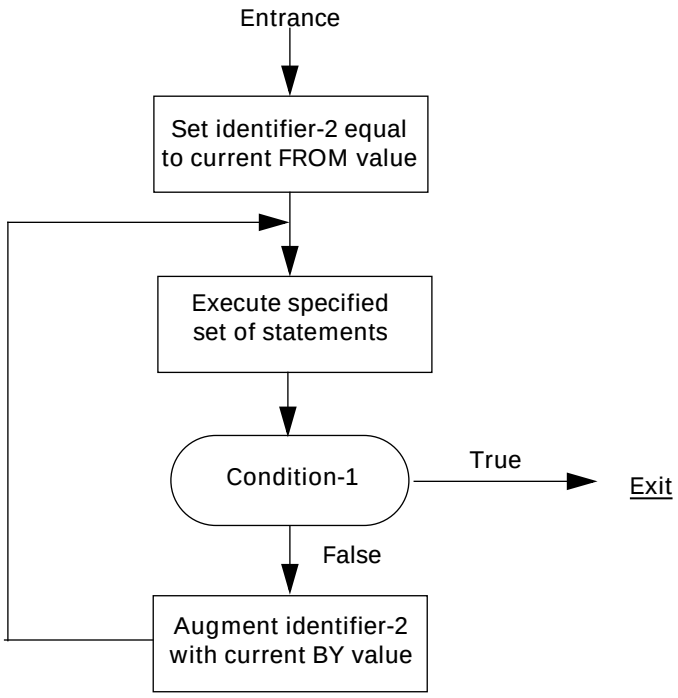


Figure C.13 — The VARYING phrase of a PERFORM statement with the TEST AFTER phrase having one condition

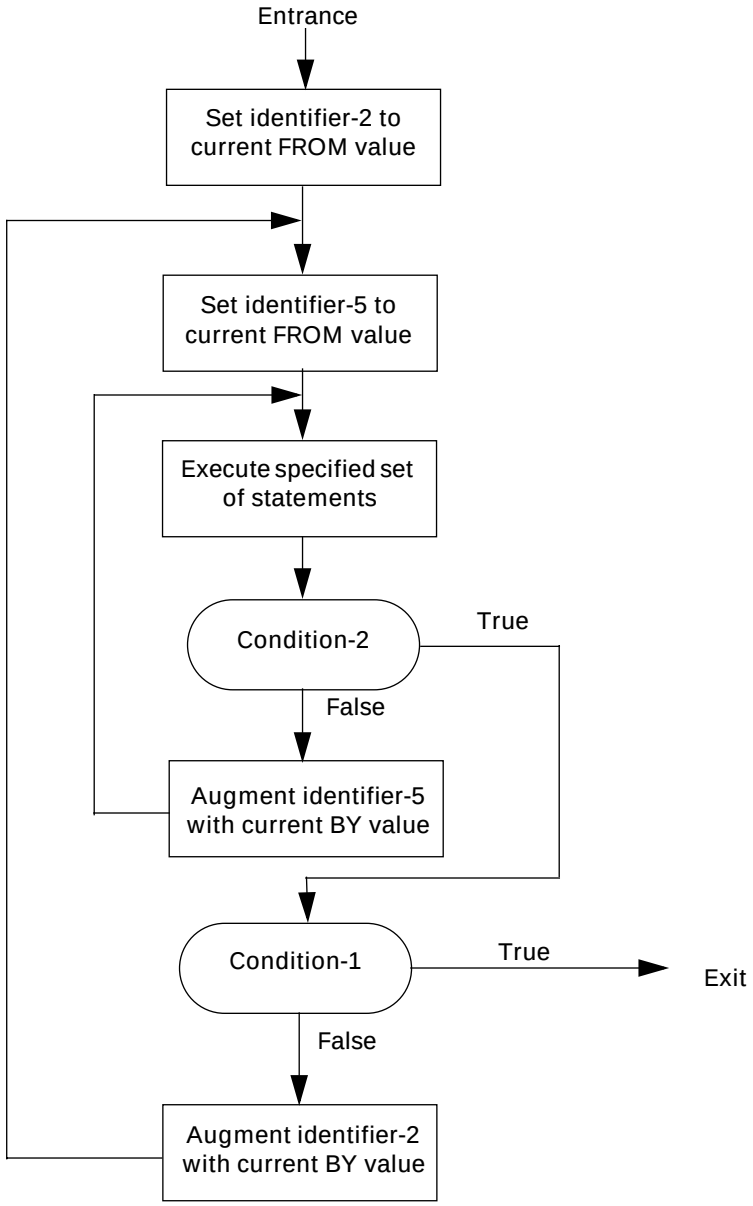


Figure C.14 — The VARYING phrase of a PERFORM statement with the TEST AFTER phrase having two conditions

D.25 Example of free-form reference format

An example of free-form reference format follows. It is assumed that each line except the first one begins in column 1, but they can begin in any column. The first line begins in column 8.

```
       >>SOURCE FORMAT IS FREE
IDENTIFICATION DIVISION.
PROGRAM-ID.  Free-form-example.
SOURCE-COMPUTER.  xyz.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  a-long-item PIC X(100) VALUE "The first part continued on the next "-
        "line - simple enough.".
01  num-item PIC 9(10).
PROCEDURE DIVISION.
a-paragraph-name.
*> This is a comment line - nothing precedes the comment indicator
    MOVE 0 TO num-item *> this is an inline comment - text precedes
    MOVE "a literal that is concatenated to make a " &
         "longer one" TO a-long-item
.   *> a period still has to end a paragraph
>>SOURCE FORMAT IS FIXED
           *> Now we are in fixed format - wasted space at left
       b-paragraph-name.
           MOVE 1 TO num-item   *> you can use inline comments
       >>SOURCE FORMAT IS FREE
*> now we are back in free-form - note the directive is in column 8
.
c-paragraph-name.
    ADD 1 TO num-item *> note that indentation not needed - there is no area a
    *> or area b or any area
    MOVE "abc" TO a-long-item *> however, it makes a better program
    STOP RUN
    .
```

D.26 Conditional compilation

Conditional compilation provides a means of including or omitting selected lines of source code depending on the values of literals specified by the DEFINE directive. In this way, multiple variants of the same program may be created, without the need to maintain separate source streams.

The compiler directives that are used for conditional compilation are the DEFINE directive, the EVALUATE directive, and the IF directive. The DEFINE directive is used to define constants that are referenced in the EVALUATE and IF directives in order to select lines of code that are to be compiled or are to be omitted during compilation.

The following examples illustrate how conditional compilation is used:

```
>>DEFINE compile-this AS 1
>>DEFINE system-number AS 14
>>DEFINE system-type AS "type A"
IDENTIFICATION DIVISION.
PROGRAM-ID.  A-program.
DATA DIVISION.
>>EVALUATE system-type
  >>WHEN "type A"
    01 type-item PIC X(10) VALUE "System A".
  >>WHEN "type B"
    01 type-item PIC X(10) VALUE "System B".
  >>WHEN OTHER
    01 type-item PIC X(10) VALUE "Bad system".
>>END-EVALUATE
    ...
PROCEDURE DIVISION.
Startt.
>>IF compile-this IS DEFINED
   DISPLAY "something"          *> compiled only when compile-this is specified
>>END-IF
  ...
>>IF compile-this IS NOT DEFINED
   DISPLAY "something else"     *> compiled only when compile-this is not specified
>>END-IF
>>IF (compile-this = 1 AND system-type = "type A") OR (compile-this = 2 AND system-type = "type Q")
    PERFORM something
>>ELSE
    PERFORM something-else
>>END-IF
>>IF system-number IS > 10 AND system-number IS < 20
    MOVE x TO y
>>ELSE
    MOVE z TO y
>>END-IF

 ...
```

D.27 CALL-CONVENTION directive

The CALL-CONVENTION directive indicates how a program-name or a method-name specified as a literal in a subsequent INVOKE statement, inline method invocation, CALL statement, CANCEL statement, or program-address-identifier is to be processed by the compiler. It may also be used by the implementor to indicate other aspects of the call. The default is as if CALL-CONVENTION IS COBOL had been specified. If call-convention-name-1 is specified, an implementor-defined convention is used.

In the following example, if call-convention-name-1 indicates C names, all names without AS clauses are treated as C names and are case-sensitive:

```
>>CALL-CONVENTION IS COBOL           *> case-insensitive
     CALL "A-COBOL-PROGRAM" USING

>> CALL-CONVENTION IS c-type-names  *> case-sensitive
     CALL "A_C_program" USING
   ...
>> CALL-CONVENTION IS COBOL
*> now we are back to COBOL names again
```

D.28 ENTRY-CONVENTION clause

The ENTRY-CONVENTION clause specifies how a runtime element is to receive control. It may be specified in the OPTIONS paragraph of a function or program, indicating the manner by which the containing runtime element will receive control. It may be specified in the OPTIONS paragraph of a class, indicating the manner by which the methods of that class will receive control. It may also be specified in the OPTIONS paragraph of an interface or prototype, to allow proper transfer of control to the methods, functions, or programs referenced by those interfaces or prototypes.

The effect of the ENTRY-CONVENTION clause is implementor-defined and may include such aspects of the implementation as the calling sequence used, the manner in which the stack is set up, and any transformation of the called program name.

D.29 Date and time handling

COBOL provides a wide range of mechanisms associated with date and time handling. These mechanisms have evolved over the years of COBOL's existence and make use of a number of capabilities of the language.

For purposes of illustration, all date examples in this section will use a date of Wednesday, February 15, 1995, and all time examples will use 05:14:27.812479168304 Eastern Standard Time (on the same calendar date).

These mechanisms use various terms to describe internal formats, and the definitions are paraphrased here for clarification:

1) Standard numeric date form: a date in the Gregorian calendar in the form YYYYMMDD, in which the first four character positions represent the year, the next two the month of the year, and the next two the date of the month. The example date above is represented as 19950215 in standard numeric date form.

2) Standard numeric time form: a numeric time value representing seconds past midnight. The mechanisms that accept this form require that the implementation be able to recognize at least nine digits to the right of the decimal point in the value, thus providing the ability to represent times to nanosecond precision. (The implementor is not required to provide the current time in the operating environment to nanosecond accuracy, but is encouraged to provide as accurate a value as feasible in that form). The example time above is represented in standard numeric time form as 18867.812479168304.

3) Integer date form: an integer value representing the number of days a given date succeeds December 31, 1600 in the Gregorian calendar. The example date above is represented in integer date form as 143951.

D.29.1 ACCEPT

The first mechanism COBOL provided for date and time handling was through the ACCEPT statement and the use of conceptual data items through which the values were provided by the operating environment. This capability was originally standardized in ANSI X3.23-1974 with DATE, DAY and TIME as special registers, was enhanced to include DAY-OF-WEEK in ISO 1989:1985, and further enhanced to include DATE YYYYMMDD and DAY YYYYDDD special registers by ISO/IEC 1989:2002.

All of these special registers reflect the current date and time as provided by the operating environment.

D.29.1.1 ACCEPT FROM DATE

ACCEPT FROM DATE provides a six-digit usage display numeric result representing the current Gregorian date: two digit year of century, two digit month, and two digit day of month; for example, 950215.

D.29.1.2 ACCEPT FROM DATE YYYYMMDD

ACCEPT FROM DATE YYYYMMDD provides an eight-digit usage display numeric result representing the current Gregorian date, in standard numeric date form; for example, 19950215.

D.29.1.3 ACCEPT FROM DAY

ACCEPT FROM DAY provides a five-digit usage display numeric result representing the current Julian date: two digit year of century, and three digit day of the year in the range 001 through 366; for example, 95046.

D.29.1.4 ACCEPT FROM DAY YYYYDDD

ACCEPT FROM DAY YYYYDDD provides a seven-digit usage display numeric result representing the current Julian date: four digit year, and three digit day of the year in the range 001 through 366; for example, 1995046.

D.29.1.5 ACCEPT FROM DAY-OF-WEEK

ACCEPT FROM DAY-OF-WEEK provides a one digit usage display numeric result representing the current day of the week, where the value 1 represents Monday, 2 represents Tuesday, ... and 7 represents Sunday; for example, 3.

D.29.1.6 ACCEPT FROM TIME

ACCEPT FROM TIME provides an eight digit usage display numeric result representing the current local time based on a 24-hour clock: two digits for the hour past midnight in the range 00 through 23; two digits for the minutes past the hour in the range 00 through 59; two digits for the seconds past the minute in the range 00 through 59 by default; and two digits for the hundredths of the second past the second; for example, 05142781.

In ISO/IEC 1989:2002, this mechanism was enhanced so that the content of the seconds subfield is affected by the current setting of the LEAP-SECOND directive. If the directive with the ON phrase is in effect, the range of the seconds subfield is defined by the implementor; otherwise, the maximum value is 59.

D.29.2 Basic intrinsic functions

ISO 1989:1985/Amd 1, an amendment to ISO 1989:1985, added a number of intrinsic functions for date and time handling in COBOL. These intrinsic functions provided not only enhanced operating environment date and time retrieval mechanisms but also functions designed to facilitate the manipulation and conversion of dates.

Two additional functions were added by ISO/IEC 1989:2002 as complements to these basic functions.

The order in which these functions are presented here is intended to illustrate the interrelationships among them.

D.29.2.1 CURRENT-DATE function

The CURRENT-DATE function returns a 21-character alphanumeric value representing the local date and time in the operating environment as follows:

- Four numeric digits of the year in the Gregorian calendar
- Two numeric digits of the month of the year
- Two numeric digits of the day of the month
- Two numeric digits of the hour past midnight
- Two numeric digits of the minutes past the hour
- Two numeric digits of the seconds past the minute
- Two numeric digits of the hundredths of a second past the minute
- The character "+" to indicate that local time is the same as, or ahead of, Greenwich Mean Time, "-" to indicate that local time is behind Greenwich Mean Time, or "0" to indicate that the operating environment does not have the facility to provide the local time differential
- Two digits indicating the number of hours the reported time is ahead of or behind Greenwich Mean Time
- Two digits indicating the number of additional minutes the reported time is ahead of or behind Greenwich Mean Time

For example, in operating environments that include the ability to differentiate local time from Greenwich Mean Time, the returned result would be "1995021505142781-0500" ; in operating environments that do not have that capability, the returned result would be "19950215051427810000".

D.29.2.2 WHEN-COMPILED function

The WHEN-COMPILED function returns a value in exactly the same format as the CURRENT-DATE function, representing the local date and time in the operating environment at which the compilation of the program occurred.

D.29.2.3 INTEGER-OF-DATE function

The INTEGER-OF-DATE function accepts an integer parameter representing a date in the Gregorian calendar in standard numeric date form (YYYYMMDD), and returns a positive integer in integer date form (days past December 31. 1600); for example, given the parameter 19950215, the returned value would be 143951.

D.29.2.4 INTEGER-OF-DAY function

The INTEGER-OF-DAY function accepts an integer parameter in Julian date form (YYYYDDD) and returns a positive integer in integer date form (days past December 31, 1600); for example, given the parameter 1995046, the returned value would be 143951.

D.29.2.5 DATE-OF-INTEGER function

The DATE-OF-INTEGER function performs the reverse of the INTEGER-OF-DATE function. Given a value in integer date form (days past December 31, 1600), the function returns a positive integer in the form YYYYMMDD; for example, for the value 143951, the returned value would be 19950215.

D.29.2.6 DAY-OF-INTEGER function

The DAY-OF-INTEGER function performs the reverse of the INTEGER-OF-DAY function. Given a value in integer date form (days past December 31, 1600), the function returns a positive integer in the form YYYYDDD; for example, for the value 143951, the returned value would be 1995046.

D.29.2.7 TEST-DATE-YYYYMMDD function

The TEST-DATE-YYYYMMDD function, introduced by ISO/IEC 1989:2002, accepts an argument in standard date form (YYYYMMDD). It returns a zero if the argument is a valid date; the value 1 if the year subfield content is out

of range; the value 2 if the month subfield content is out of range; or the value 3 if the day subfield content is out of range for the given year and month.

D.29.2.8 TEST-DAY-YYYYDDD function

The TEST-DAY-YYYYDDD function, introduced by ISO/IEC 1989:2002, accepts an argument in Julian date form (YYYYDDD). It returns a zero if the argument is a valid date; the value 1 if the year subfield content is out of range; or the value 2 if the day subfield content is out of range for the given year.

D.29.3 Locales

With the introduction of Locales in ISO/IEC 1989:2002, three additional intrinsic functions were provided to support conversion of dates and times from standard numeric forms into locale-specific forms. The previous COBOL standard added a fourth intrinsic function associated with Locales.

D.29.3.1 LOCALE-DATE function

The LOCALE-DATE function takes as its first argument a date in standard date form (YYYYMMDD) and an optional second argument specifying a locale identified in the SPECIAL-NAMES paragraph (if this argument is omitted, the current locale is used), and returns a character string containing the date specified by the first argument in the format associated with the locale.

D.29.3.2 LOCALE-DAY function

The LOCALE-DAY function takes as its first argument a date in Julian date form (YYYYDDD) and an optional second argument specifying a locale identified in the SPECIAL-NAMES paragraph (if this argument is omitted, the current locale is used), and returns a character string containing the date specified by the first argument in the format associated with the locale.

D.29.3.3 LOCALE-TIME function

The LOCALE-TIME function takes as its first argument an integer time in HHMMSS form and an optional second argument specifying a locale identified in the SPECIAL-NAMES paragraph (if this argument is omitted, the current locale is used), and returns a character string containing the time specified by the first argument in the format associated with the locale.

D.29.3.4 LOCALE-TIME-FROM-SECONDS function

The LOCALE-TIME-FROM-SECONDS is introduced in this final committee draft International Standard. This function accepts as its first argument a value in standard numeric time form and an optional second argument specifying a locale identified in the SPECIAL-NAMES paragraph (if this argument is omitted, the current locale is used), and returns a character string containing the time specified by the first argument in the format associated with the locale. Note that while the LOCALE-TIME function does not provide for the representation of fractional seconds, the LOCALE-TIME-FROM-SECONDS function must be able to recognize and process argument values representing precision to the nanosecond (the implementor may allow even more precise values).

D.29.4 ISO 8601 date and time format handling

Intrinsic functions to support the handling of dates and times in some of the formats specified in ISO 8601, Representations of Dates and Times, are introduced in this final committee draft International Standard.

For purposes of illustration, the examples will use three formats: the basic calendar date format ("YYYYMMDD"); a basic local time format with two digits of fractional seconds and offset from GMT ("hhmmss.ss+hhmm"), and a combined date-and-time format that uses these two formats ("YYYYMMDDThhmmss.ss+hhmm").

The order in which these functions are presented here is intended to illustrate the interrelationships among them.

D.29.4.1 FORMATTED-CURRENT-DATE function

The FORMATTED-CURRENT-DATE function has as its single argument a literal containing a combined date and time format, and returns the local date and time in the operating environment in the specified format. For example, given the format "YYYYMMDDThhmmss.ss+hhmm" and a current operating environment date as described in the introductory paragraphs of D.29, Date and time handling, the returned value will be "19950215T05142781-0500".

NOTE Given this particular format – "YYYYMMDDThhmmss.ss+hhmm" – the only difference between the returned value for CURRENT-DATE and that for FORMATTED-CURRENT-DATE is the presence of the character "T" separating the date portion from the time portion in the returned value of the latter function.

D.29.4.2 SECONDS-PAST-MIDNIGHT function

The SECONDS-PAST-MIDNIGHT function has no parameters. It returns a value in standard numeric time form representing the current time of day in the operating environment expressed as seconds past midnight. For example, if the current time in the operating environment is 5:14:27.812479168304, the returned value would be as close to 18867.812479168304 as the operating environment was capable of providing.

D.29.4.3 FORMATTED-DATE function

The FORMATTED-DATE function accepts two arguments -- a literal that is a valid date format, and a value in integer date form -- and returns a date in the specified format. For example, for the format "YYYYMMDD " and the value 143951, the returned value would be 19950215.

D.29.4.4 FORMATTED-TIME function

The FORMATTED-TIME function accepts three arguments – a literal that is a valid time format; a value in numeric time form; and an optional integer argument specifying the number of minutes by which the second argument is presumed to differ from GMT – and returns a time in the specified format. If the first parameter is either a UTC format or an offset format, the third parameter is required (although it may be zero). For example: given as the first argument the format "hhmmss.ss+hhmm", as the second argument the value 18867.812479168304 representing local time, and as the third argument the value +300 representing the five hours by which Eastern Standard Time differs from UTC, the returned value would be "05142781+0500".

D.29.4.5 FORMATTED-DATETIME function

The FORMATTED-DATETIME function is a combination of the FORMATTED-DATE and FORMATTED-TIME functions. It accepts four arguments – a literal that is a valid combined date and time format; a value in integer date form; a value in standard numeric time form; and an optional offset parameter (see the FORMATTED-TIME function) – and returns a combined date and time in the specified format. For example, given as the first argument the format "YYMMDDThhmmss.ss+hhmm", as the second argument the value 143951, as the third argument the value 18867.812479168304, and as the fourth argument the value +300, the returned value would be "19950215T05142781+0500".

D.29.4.6 INTEGER-OF-FORMATTED-DATE function

The INTEGER-OF-FORMATTED-DATE function accepts two parameters – a literal that is either a valid date format or a valid combined date and time format, and a data item whose content is in the specified format – and returns a value in integer date form. If the first argument is a combined date and time format, the time portion of the data item is validated against the format, but the returned value does not include the time. For example, given as the first argument the format "YYYYMMDD" and as the second argument the value "19950215", the returned value would be 143951. The same value would be returned if the first argument was the format "YYYYMMDDThhmmss.ss+hhmm" and the second argument the value "19950215T05142781+0500".

D.29.4.7 SECONDS-FROM-FORMATTED-TIME function

The SECONDS-FROM-FORMATTED-TIME function accepts two parameters – a literal that is either a time format or a combined date and time format, and a data item whose content is in the specified format – and returns a value in standard numeric time form. If the first argument is a combined date and time format, the date portion of the

data item is validated against the format, but the returned value does not include the date. For example, given as the first argument the format "hhmmss.ss+hhmm" and as the second argument the value "05142781+0500", the returned value would be 18867.81. The same value would be returned if the first argument was the format "YYYYMMDDThhmmss.ss+hhmm" and the second argument was the value "19950215T05142781+0500".

D.29.4.8 TEST-FORMATTED-DATETIME function

The TEST-FORMATTED-DATETIME function accepts two parameters – a date format, a time format, or a combined date and time format, and a data item containing a data item presumed to be in that format. The function returns zero if the data in the second argument exactly matches the format in the first argument. If the data item does not match the format, the value returned is the ordinal position of the first character for which the data item and the format do not match.

D.29.4.9 COMBINED-DATETIME function

The COMBINED-DATETIME function accepts two arguments – a date in integer date form, and a time in standard numeric time form – and returns a numeric value in which the date occupies the integer part of the value and the time represents the fractional part, according to the expression argument-1 + (argument-2 / 100000). For example, given the integer date form value 143951 (representing the date February 15, 1995) and the standard numeric time form value 18867.812479168304 (representing the time 05:14:27.812479168304), the returned value would be exactly 143951.18867812479168304. The intent of this function is to provide a standard numeric representation of a combined date and time in a single variable, for any date from January 1, 1601 onward, to at least nanosecond precision.

Annex E
(informative)

Substantive changes list

E.1 General

This annex contains a list of the substantive changes between the previous COBOL standard and this final committee draft International Standard. The list is separated into those changes potentially affecting existing COBOL programs and those changes not affecting existing COBOL programs.

E.2 Substantive changes potentially affecting existing programs

1) **Obsolete elements.** The following features that were classified as obsolete in the previous COBOL standard, have been removed from this final committee draft International Standard:

   — Communication facility
   — Debugging lines and the DEBUGGING MODE phrase
   — PADDING CHARACTER clause

   Justification:

   In each case, the justification for eventually removing the feature was presented in the previous COBOL standard. These features are either ones better handled by external hardware or software, or are irrelevant in a modern development environment.

   It is believed that several of these features were infrequently implemented and that few programs have used these features. For those users that remain dependent on these features, implementors may provide extensions that correspond to the deleted features.

2) **Reserved words.** The following reserved words have been added:

FLOAT-BINARY-7	FUNCTION-POINTER
FLOAT-BINARY-16	INFINITY
FLOAT-BINARY-34	REPRESENTS-NOT-A-NUMBER
FLOAT-DECIMAL-16	<>
FLOAT-DECIMAL-34	

   Justification:

   In each case, the benefits to be derived from the additional facility provided through the addition of each reserved word were deemed to outweigh the inconvenience caused by removing this word from the realm of user-defined words.

3) **ARITHMETIC IS STANDARD.** This previously required feature is now classified as processor-dependent and obsolete.

   Justification:

   The standard arithmetic feature of the previous COBOL standard provided a mechanism whereby some arithmetic operations and functions produced portable results. The intermediate data item used for these computations was an abstract concept in the previous COBOL standard; there was no corresponding phrase in the USAGE clause to allow the declaration of items in a format conforming to the intermediate data item.

The STANDARD-DECIMAL phrase of the ARITHMETIC clause specifies that intermediate results be produced in a format that is described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic. The FLOAT-DECIMAL-34 phrase of the USAGE clause provides for a portable representation of data in that same format. The combination of these two features provides capabilities that exceed those of standard arithmetic in the previous COBOL standard.

It is believed that few if any programs will be impacted by this change because there are no known implementations of ARITHMETIC IS STANDARD.

4) **New intrinsic functions.** The intrinsic functions listed below have been introduced in this final committee draft International Standard. These names may no longer be used as user-defined words within the scope of a REPOSITORY paragraph that includes the function-specifier FUNCTION ALL INTRINSIC:

COMBINED-DATETIME
FORMATTED-CURRENT-DATE
FORMATTED-DATE
FORMATTED-DATETIME
FORMATTED-TIME
INTEGER-OF-FORMATTED-DATE
LOCALE-TIME-FROM-SECONDS
SECONDS-FROM-FORMATTED-TIME
SECONDS-PAST-MIDNIGHT
TEST-FORMATTED-DATETIME
TRIM

Justification:

New intrinsic functions were added to provide needed functionality. Existing programs that include a REPOSITORY paragraph with the intrinsic format of the function-specifier with the ALL phrase specified may no longer compile if any of these names are specified as user-defined words within the scope of the REPOSITORY paragraph.

The benefits to be derived from the additional capabilities provided through the addition of each intrinsic function name were deemed to outweigh the inconvenience caused by removing this word from the realm of user-defined words available to the user in the limited context of the scope of a REPOSITORY paragraph with the FUNCTION ALL INTRINSIC specification

5) **General case mappings.** The list of mapping of uppercase letters to lowercase letters in the COBOL character repertoire has been changed.

Deleted pair

(<U0256>,<U0189>)

Added pairs

(<U01F6>,<U0195>);(<U01F7>,<U01BF>);(<U01F8>,<U01F9>);(<U0218>,<U0219>);
(<U021A>,<U021B>);(<U021C>,<U021D>);(<U021E>,<U021F>);(<U0222>,<U0223>);
(<U0224>,<U0225>);(<U0226>,<U0227>);(<U0228>,<U0229>);(<U022A>,<U022B>);
(<U022C>,<U022D>);(<U022E>,<U022F>);(<U0230>,<U0231>);(<U0232>,<U0233>);
(<U03DA>,<U03DB>);(<U03DC>,<U03DD>);(<U03DE>,<U03DF>);(<U03E0>,<U03E1>);
(<U0400>,<U0450>);(<U040D>,<U045D>);(<U048C>,<U048D>);(<U048E>,<U048F>);
(<U04EC>,<U04ED>);(<U0554>,<U0584>);(<U0555>,<U0585>);(<U0556>,<U0586>);
(<U2126>,<U03C9>);(<U212A>,<U006B>);(<U212B>,<U00E5>);(<U2160>,<U2170>);
(<U2161>,<U2171>);(<U2162>,<U2172>);(<U2163>,<U2173>);(<U2164>,<U2174>);
(<U2165>,<U2175>);(<U2166>,<U2176>);(<U2167>,<U2177>);(<U2168>,<U2178>);
(<U2169>,<U2179>);(<U216A>,<U217A>);(<U216B>,<U217B>);(<U216C>,<U217C>);
(<U216D>,<U217D>);(<U216E>,<U217E>);(<U216F>,<U217F>)

Justification:

These changes were made to align with UnicodeData.txt for Unicode 5.2.0. The case mapping table in the previous standard was extracted from ISO/IEC DTR 14652, but this technical report was withdrawn. Because there are no known implementations of these case mappings, it is believed that no existing programs will be affected by this change.

6) **Characters permitted in user-defined words.** The following special characters, represented in the hexadecimal code values that identify letters specified in ISO/IEC 10646, have been deleted:

— 2118 SCRIPT CAPITAL P
— 212E ESTIMATED SYMBOL

Justification:

Those two characters were listed in ISO/IEC TR 10176:2001 but are no longer listed in ISO/IEC TR 10176:2003.

It is believed that few, if any, programs will be impacted by this change because there are no known implementations of the previous COBOL standard that included those characters in their COBOL character repertoire. ISO/IEC TR 10176:2003 represents a refinement of ISO/IEC TR 10176:2001. The benefits of maintaining currency in references to external standards outweigh the inconvenience caused by removing those special characters from the list of the characters permitted in user-defined words.

7) **CURRENCY SIGN clause, equivalent currency symbols with different currency strings disallowed:** No two CURRENCY SIGN clauses may specify the same currency symbol, or currency symbols that are equivalent, unless the currency strings associated with the CURRENCY SIGN clauses are identical. Equivalence among basic letters, extended letters, and basic special characters is determined according to the general rules for the COBOL character repertoire. Equivalence determination for other characters is defined by the implementor.

Justification:

When two or more currency symbols were associated with different currency strings, it was undefined which currency string would appear in data.

It is believed that few programs will be affected by this change.

8) **CURRENCY SIGN clause, hexadecimal literal restrictions:** A hexadecimal literal can only be specified as a currency string if the PICTURE SYMBOL phrase is specified in the associated CURRENCY SIGN clause. A hexadecimal literal shall not be specified as a currency symbol.

Justification:

The meaning of a hexadecimal literal is determined at run time. The prohibition against the use of a hexadecimal literal as a currency symbol, and the restriction on the use of a hexadecimal literal as a currency string, are necessary because the meaning of the currency symbol must be known at compile time.

It is believed that few programs will be affected by these changes.

9) **Extended letters made optional.** Extended letters in the COBOL character repertoire, which were required in the previous standard, are made optional in this committee draft International Standard.

Justification:

This change allows implementation on systems with limited capability.

It is expected that no users will be impacted by this change because there are no known implementations of extended letters as of this writing. Implementors can continue to provide extended letters.

10) **Required features made optional**. A number of required features of the previous COBOL standard have been made optional in this final committee draft International Standard.

    Justification:

    The features in this category are those for which there is strong sentiment that they are either inappropriate or irrelevant for all COBOL environments and implementations. Note that in some cases the features have also been made obsolete.

    It is expected that few users will be impacted by this change because implementors are free to continue to provide these optional features.

    The features that were required in the previous COBOL standard and made optional in this final committee draft International Standard are presented in more detail in A.4, Optional language element list, as follows:

    A.4.1, ACCEPT and DISPLAY screen handling

    A.4.3, ARITHMETIC IS STANDARD

    A.4.4, COBOL character repertoire

    A.4.7, File sharing and record locking

    A.4.10, Locale support and related functions

    A.4.11, Object orientation

    A.4.12, Report Writer

    A.4.14, VALIDATE

11) **MOVE statement, same sending and receiving operands, alphanumeric-edited.** The result of a MOVE statement in which the sending operand is of category alphanumeric-edited, the sending operand and the receiving operand are described by the same data-description entry and are found by item identification to occupy the same storage area, is undefined.

    Justification:

    The previous COBOL standard specified that all such cases produced defined results if no specific rule stated otherwise, and there was no such explicit statement in the case of MOVE. Therefore, in the previous standard, the results of such a MOVE statement were specified as defined. However, the rules in that previous standard did not describe the result in an unambiguous way. For that reason, the result in this case was not defined. In this final committee draft International Standard, the result of such a statement is explicitly described as undefined.

    It is believed that few programs will be affected by this change.

12) **MOVE statement, same sending and receiving operands, variable-length.** The result of a MOVE statement in which the sending operand and the receiving operand are variable-length data items, the size of the operand is determined by a data item subordinate to the operand, and the size of the operand is less than the upper limit specified for the OCCURS clause in the table declaration subordinate to the operand, is the same as if the information in the sending operand had been moved to a fixed-length temporary field of the same class, category, and length as the sending operand, and the content of that temporary then moved to the receiving operand.

    Justification:

It was not clear from the rules for overlapping operands, for OCCURS, and for MOVE as to whether the result was undefined (as the overlapping operand rule would have it) or well-defined as producing space fill (as OCCURS might be understood to indicate), but having the result of a statement be explicitly defined by the standard makes the statement more useful than leaving the result undefined or explicitly describing it as undefined.

It is believed that few programs will be affected by this change.

13) **Overlapping operands, defined and undefined results.** When a sending and a receiving data item in any statement share a part or all of their storage areas and are defined by the same data description entry, and the rules for the statement, or the sequence of operations specified by the rules for the statement, provide for a well-defined result, then the result is defined by the rules for the statement. The overlapping operand rule at 14.6.9, Overlapping operands, now applies only when no rules for the statement can be applied to the result. Previously it was considered to override any rules that could be applied to provide a defined result.

Justification:

This resolves an inconsistency in the previous COBOL standard in which the rules for some statements were clear as to what the result would be in such a case and yet the application of the overlap rule made them undefined. In some cases (for CALL and COMPUTE, for example), the rules for the statement were clear that the information in the sending operands was identified and retrieved before any process could affect the receiving operands; thus, the content of the receiving operands was well-defined even when that same process affected the sending operands. The rules for each statement have been reviewed.

It is believed that few programs will be affected by this change.

14) **ACTIVE-CLASS for object property and inline method invocation.** Use of an object reference described with the ACTIVE-CLASS phrase is now prohibited for returning items of get property methods, parameters of set property methods, and returning items for inline method invocations.

Justification:

In the previous standard, the ACTIVE-CLASS in the returning item of a get property method, a parameter to a set property method, or the returning item in an inline method invocation could be different from the ACTIVE-CLASS in the activating runtime element. In these circumstances, a runtime conformance error could occur. The change prevents that possibility.

It is believed that few programs will be affected by this change.

15) **STANDARD-COMPARE function, the default ordering table has changed from 'ISO14651_2002_TABLE1' to 'ISO14651_2006_TABLE1'.** The new table contains several additional characters which may affect the result of some comparisons.

Justification:

The table name was updated in the new version of ISO/IEC 14651.

16) **Processor-dependence on ISO/IEC 14651.** When the processor-dependent feature for ordering standard ISO/IEC 14651 is not supported, the implementor is no longer required to accept the syntax of the STANDARD-COMPARE intrinsic function, the EC-ORDER-NOT-SUPPORTED exception condition, and the ORDER TABLE clause in the SPECIAL-NAMES paragraph.

Justification:

Because support for ordering standard ISO/IEC 14651 is processor-dependent, the acceptance of the associated syntax and the setting of the EC-ORDER-NOT-SUPPORTED exception condition is an anomaly that does not merit the implementation cost.

It is expected that no users will be impacted by this change because there are no known implementations of COBOL support for ISO/IEC 14651 as of this writing.

E.3 Substantive changes not affecting existing programs

1) **ADDRESS OF.** The ADDRESS OF phrase now allows FUNCTION keyword.

2) **Any-length elementary items.** PREFIXED and DELIMITED phrases in the ANY LENGTH clause provide the ability to describe the representation of each any-length data item.

3) **Characters permitted in user-defined words.** 9470 more characters are added as the characters permitted in user-defined words; the following characters, represented in the hexadecimal code values that identify letters specified in ISO/IEC 10646, are no longer special characters.

— 093D DEVANAGARI SIGN AVAGRAHA
— 0B3D ORIYA SIGN AVAGRAHA

4) **Characters permitted in text-words.** In order to allow manipulation by COPY…REPLACING and REPLACE statements of source text and library text containing currency symbols that are characters from outside the COBOL character repertoire, characters from outside that repertoire are now allowed in, or as, text-words.

5) **COBOL character repertoire, equivalence of characters in literals.** There are cases in which a literal contains a letter or letters whose value has an effect on compilation (for example, a currency symbol), and the rules for the context of that literal allow uppercase-to-lowercase equivalence as well as basic-to-extended-letter equivalence. The rule that no case mapping occurs for the non-hexadecimal formats of alphanumeric and national literals has been relaxed to allow for such exceptions.

6) **Currency symbol and character mapping.** A currency symbol that is a basic letter, an extended letter, or a basic special character from within the COBOL character repertoire is equivalent to another currency symbol if the two characters are equivalent according to the general rules for the COBOL character repertoire. For currency symbols other than basic letters, extended letters, and basic special characters, equivalence is defined by the implementor. This affects the determination of uniqueness of currency symbols among multiple CURRENCY-SIGN clauses, as well as the identification of currency symbols within PICTURE character-strings.

7) **Exception conditions.** New exception conditions have been added covering cases where the capacity of a dynamic-capacity table exceeds the maximum or is below the minimum capacity (EC-BOUND-OVERFLOW, EC-BOUND-SET, EC-BOUND-TABLE-LIMIT), or where the capacity is changed during a SEARCH (EC-FLOW-SEARCH), or where a dynamic-capacity table or any-length data item is found to be corrupt (EC-DATA-INTEGRITY).

8) **Function-identifier.** The function-identifier now allows specification of a function pointer.

9) **IEEE floating-point numeric handling.** The ability to perform basic arithmetic with intermediate results maintained in Decimal128 and Binary128 formats as described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, has been added.

10) **IEEE floating-point usages.** The ability to define data items in Decimal64, Decimal128, Binary32, Binary64 and Binary128 formats as specified in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, has been added.

11) **INITIALIZE statement.** The syntax has been enhanced to allow the specification of multiple categories in a single REPLACING phrase of an INITIALIZE statement.

12) **LOWER-CASE function.** A locale can be specified in the argument list to identify a locale for use in mapping uppercase to lowercase letters.

13) **Maximum size of a text-word in pseudo-text and library-text.** The maximum size of text-words in pseudo-text and in library text has been increased from 322 character positions to 65,535 character positions.

14) **Maximum size of literals.** The maximum sizes of alphanumeric, boolean, and national literals (including those formed using concatenation expressions) have been increased from 160 character positions to 8,191 character positions.

15) **Method overloading.** Within a class, methods with the same name will no longer generate an error if the number of parameters is different, or the parameters are of different types or lengths.

16) **OCCURS clause.** A new format that allows the definition of a dynamic-capacity table has been introduced. Entries in such a table can be created or deleted dynamically.

17) **PICTURE character repertoire.** The set of characters allowed in a PICTURE character-string is expanded to include currency symbols from outside the COBOL character repertoire.

18) **READ file-name-1 INTO identifier-1.** A storage area identified by identifier-1 may now be the same storage area as, or be shared with, the record area associated with file-name-1; they are no longer prohibited from being the same storage area. In the event these storage areas partially or completely overlap, the rules specifying whether the results are defined or undefined are those of the MOVE statements as applied to the implicit MOVE described in the general rules for READ.

19) **Record locks.** If the SHARING clause in the file control entry and the SHARING phrase on the OPEN statement are supported by the processor, the implementor shall support a minimum of 15 simultaneous record locks for a given file connector, and a minimum of 255 simultaneous record locks within a given run unit.

20) **Relational operator enhancement.** The symbol <> has been added as an extended relational operator. It is an abbreviation for 'not equal'.

21) **RELEASE record-name FROM identifier-1.** It is no longer prohibited for Identifier-1 and record-name-1 to reference the same storage area. In the event the storage areas identified by identifier-1 and record-name-1 partially or completely overlap, the rules specifying whether the results are defined or undefined are those of the MOVE statement as applied to the implicit MOVE described in the general rules for RELEASE.

22) **RETURN file-name-1 INTO identifier-1.** A storage area identified by identifier-1 may now be the same storage area as the record area associated with file-name-1; they are no longer prohibited from being the same storage area. In the event a part or all of these storage areas are shared, the rules specifying whether the results are defined or undefined are those of the MOVE statement as applied to the implicit MOVE described in the general rules for RETURN.

23) **REWRITE record-name-1 FROM identifier-1.** It is no longer prohibited for identifier-1 and record-name-1 to reference the same storage area. In the event the storage areas identified by identifier-1 and record-name-1 partially or completely overlap, the rules specifying whether the results are defined or undefined are those of the MOVE statement as applied to the implicit MOVE described in the general rules for REWRITE.

24) **Rounding enhancements.** New capabilities in intermediate and final rounding have been added. These specifications are intended to reflect rounding modes described in IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, 4, Attributes and rounding. In particular, three basic features are included: 1) the ability to specify a default rounding mode other than the preexisting nearest-away-from-zero for application in the ROUNDED clause of arithmetic statements; 2) the ability to specify a rounding mode different from the default for any given ROUNDED clause; and 3) the ability to specify rounding other than truncation during the production of any intermediate arithmetic result.

25) **SET statement.** New format has been added to enable the current capacity of a dynamic-capacity table to be altered explicitly.

26) **Structured constants.** The capability of describing a record whose content cannot be modified at run time has been added.

27) **UPPER-CASE function.** A locale can be specified in the argument list to identify a locale for use in mapping lowercase to uppercase letters.

28) **USAGE clause.** The USAGE clause may now define a function-pointer.

29) **Variable-length group enhancement.** Variable-length groups can include any-length elementary items.

30) **WRITE record-name-1 FROM identifier-1.** Identifier-1 may now reference the same storage area as record-name-1. In the event the storage areas identified by identifier-1 and record-name-1 partially or completely overlap, the rules governing whether the results are defined or undefined are those of the MOVE statement as applied to the implicit MOVE identifier-1 TO record-name-1 as described in the general rules for the WRITE statement.

31) **Zero-length literals.** A literal may now be of zero length.

Annex F
(informative)

Archaic and obsolete language element lists

F.1 Archaic language elements

The purpose of the archaic language element designation is to discourage the use in new programs of some features that are unreliable, poor programming practice, or ill-defined -- where better programming techniques are available in standard COBOL. These elements are classified as archaic rather than obsolete because their use in existing programs might be too extensive to warrant removal in the next edition of standard COBOL.

Archaic language elements are intended to remain in the next edition of standard COBOL. There is no plan for the removal of archaic elements; however, should their use in program inventories become insignificant, they may be considered for designation as obsolete by future architects of standard COBOL.

The following are archaic language elements:

1) **Continuation of COBOL words.** The continuation of COBOL words in fixed-form reference format by placing part of the word on one line and part of it on a continuation line creates programs that are difficult to read and difficult to maintain. This feature is almost never used and it does not exist in free-form reference format.

2) **MOVE of figurative constants that are not numeric (other than QUOTE and QUOTES) to numeric items.** The MOVE statement allows any figurative constant except SPACE to be moved to an integer numeric item. It allows any figurative constant to be moved to a noninteger numeric item. Because moving a figurative constant that is not numeric to a numeric data item rarely, if ever, produces a defined result (other than raising an exception) or a result that is expected by the user, the use of such a move operation should be avoided.

   The use of the new intrinsic functions HIGHEST-ALGEBRAIC and LOWEST-ALGEBRAIC permits the programmer to easily set the contents of a numeric data item to its highest or lowest possible valid value, respectively, without the error-prone task of using possibly very long numeric literals to do so. The use of these functions also makes the task of revising the length of data items in program maintenance simpler and less error prone.

   Defined results can be achieved by changing the figurative constant to a non-figurative alphanumeric literal with the exact value that is required to be moved to the numeric receiving item.

3) **NEXT SENTENCE phrase in the IF and SEARCH statements.** This phrase can be confusing, especially when specified in a delimited scope statement. It is a common belief among users that control is transferred to a position after the scope delimiter rather than to a separator period that follows it somewhere. In addition, it is a common source of errors, especially for maintenance programmers who inadvertently insert a period somewhere before the actual terminating separator period. The CONTINUE statement and scope delimiters can be used to accomplish the same functionality and such constructs are clearer and less prone to error.

4) **ON OVERFLOW phrase of the CALL statement.** This phrase is misleading because the conditions that cause the exception to occur normally are not related to an overflow condition. The only possible exception might be a memory overflow because enough memory was not available to invoke the called program, and this is very unusual or impossible in many implementations. Instead, the ON EXCEPTION phrase accomplishes the same task. That phrase is more meaningful.

5) **Identifier-n (text-n) in a COPY statement.** The use of an identifier in the REPLACING phrase of a COPY statement is misleading. The actual process inserts pseudo-text delimiters around the identifier, and it is processed as a string, not an identifier. For example, 'a IN b' will not match 'a OF b'. In another example if an identifier were defined that was the same as a function name (for example SIN) specifying 'REPLACING SIN

BY ==xyz==' would change a function call like 'FUNCTION sin (nnn)' to 'FUNCTION xyz (nnn)'. Because a large number of new formats were added to identifiers and it would be extremely difficult to parse all of them in the COPY statement, the COPY statement was changed to allow only the formats of identifiers that were allowed in the previous COBOL standard.

F.2 Obsolete language elements

Obsolete language elements are elements that will be removed in the next edition of this final committee draft International Standard because those elements are no longer needed or are rarely used. To limit the impact of removal, those elements are designated as obsolete to serve as a notice of the intention to remove them.

No elements will be removed from any future edition of this final committee draft International Standard without first having been designated as obsolete language elements in the preceding edition.

Obsolete language elements have not been enhanced or modified in this final committee draft International Standard and will not be maintained. The interaction between obsolete language elements and other language elements is undefined unless otherwise specified in this final committee draft International Standard.

A conforming implementation is required to support obsolete language elements except for elements that are also optional or processor-dependent.

The following are obsolete language elements:

1) **FLAG-85 directive.** The FLAG-85 directive was specified in the previous standard to flag incompatibilities between that standard and the one previous to it. The FLAG-02 directive is specified in the current standard to flag incompatibilities between the current standard and the previous one. There is no longer a need for the older FLAG-85 directive.

2) **ARITHMETIC IS STANDARD.** An international standard for decimal floating-point operand formats and arithmetic rules (IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, in 3.4, Binary interchange format encodings, and 3.5, Decimal interchange format encodings) specifies precision that exceeds the precision specified for the standard arithmetic feature. Implementors of other languages are planning to offer support for these formats and arithmetic rules. These factors conspire to render standard arithmetic technologically obsolete. Implementors are encouraged to provide support for ARITHMETIC IS STANDARD-DECIMAL in preference to ARITHMETIC IS STANDARD.

3) **MOVE of QUOTES to numeric items.** A MOVE of QUOTES to a numeric receiving data item will cause an EC-DATA-INCOMPATIBLE exception condition to exist and will produce undefined results. No useful purpose has been demonstrated for this capability.

Annex G
(informative)

Known errors in the standard

G.1 Rationale

While it is not the intent to publish a standard with errors, some features of COBOL have been discovered to contain errors. An attempt has been made to resolve all such matters. However, there are certain errors for which one or more of the following apply:

— Proposed solutions for the error result in unacceptable incompatibilities.

— Proposed solutions for the error are excessively complicated.

— The COBOL feature in error is archaic or obsolete.

G.2 List of errors

The following are known errors in this final committee draft International Standard:

1) **RECORD clause and I-O status 04 -- clarify READ statement, general rule 14.** General rule 14 of the READ statement currently states: "If the number of bytes in the record that is read is less than the minimum size specified by the record description entries for file-name-1, the portion of the record area that is to the right of the last valid character read is undefined. If the number of bytes in the record that is read is greater than the maximum size specified by the record description entries for file-name-1, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status is set indicating a record length conflict has occurred."

   The standard is ambiguous as to what status code is generated as a consequence of general rule 14 of the READ statement.

   The standard is ambiguous as to when and under what circumstances an I-O status 04 is returned.

   This error has not been fixed because doing so may cause unacceptable incompatibilities.

Annex H
(informative)

Bibliography

The following documents are useful references for implementors and users of this final committee draft International Standard, in addition to the normative references:

[1] *ISO/IEC Directives Part 2, Rules for the structure and drafting of International Standards, Fourth edition*, 2001, ISBN 92-67-01070-0.

[2] *Merriam-Webster's Collegiate® Dictionary, Tenth Edition*; Merriam-Webster, Incorporated, 2001, ISBN 0-87779-709-9.

[3] *The Unicode Standard, Version 5.2*, by the Unicode Consortium; ISBN 978-1-936213-00-9. The latest version can be found at the Unicode Consortium's web site, www.unicode.org, as of this writing.

[4] *ISO/IEC TR 10176:2003, Information technology - Guidelines for the preparation of programming language standards.*

Index

Symbols

"-
    Literal continuation indicator 24
& operator 128
'-
    Literal continuation indicator 24
*
    Comment line 27
* operator 128
** operator 128
*>
    Comment indicator 24
    Comment line 28
    Inline comment 28
+ operator 128
– operator 128
/
    Comment line 27
/ operator 128
:: operator 97, 128
<
    Relation 129
<=
    Relation 129
<>
    Relation 129
=
    COMPUTE statement 465
    Relation 129
    SEARCH statement 552
== pseudo-text delimiter 32, 37
>
    Relation 129
>=
    Relation 129
>>
    Compiler directive 40
    Compiler directive indicator 24

Numerics

0 PICTURE symbol 346
01 entry 111, 323
1 PICTURE symbol 344
66 RENAMES data description entry 111, 323, 366
77 level data description entry 111, 272
88 condition-name data description entry 111, 323
9 PICTURE symbol 346

A

A PICTURE symbol 341
Abbreviated combined relation conditions 154
Abnormal run unit termination 422, 424
ABS function 626
ACCEPT statement 221, 372, 397, 443
    FROM phrase 443
    Screen format 565
ACCESS MODE clause 242
    FILE-CONTROL paragraph 236
Access modes 167
ACOS function 627
Active state 417
ACTIVE-CLASS phrase 186, 439
    USAGE clause 391
ADD statement 435, 449
Addition 134
Additional language elements 5
ADDRESS OF phrase 126
    Data-address-identifier 102
    SET statement, pointer assignment 558
ADDRESS OF PROGRAM phrase
    Program-address-identifier 104
    SET statement, pointer assignment 558
Address-identifier 91
ADVANCING ON LOCK phrase
    READ statement 535
ADVANCING phrase
    READ statement 535
AFTER ADVANCING phrase
    WRITE statement 602
AFTER EXCEPTION phrase
    USE statement 593
AFTER phrase
    INSPECT statement 498
    PERFORM statement 528
AFTER STANDARD ERROR phrase
    USE statement 593
AFTER STANDARD EXCEPTION phrase
    USE statement 593
Algebraic signs 112
ALIGNED clause 285
Alignment of data
    In storage 113
    Within data items 421
ALL figurative constant 82
ALL literal 84
ALL OTHER phrase
    OPEN statement 523
    SHARING clause 254
ALL phrase
    INITIALIZE statement 493
    INSPECT statement 498
    SEARCH statement 552
    UNSTRING statement 589
ALL subscript 89
    Intrinsic function 611
ALLOCATE statement 126, 404, 452, 485
ALPHABET clause
    SPECIAL-NAMES paragraph 220

Alphabetic category 120, 344
Alphabetic character 8
Alphabetic class
Class of data
Alphabetic 120
Alphabetic data item 120
ALPHABETIC phrase
CLASS clause 149, 296
INITIALIZE statement 493
ALPHABETIC-LOWER phrase
CLASS clause 149, 296
ALPHABETIC-UPPER phrase 149
CLASS clause 296
Alphabet-name 68, 219, 400, 436, 476
CODE-SET clause 298
COLLATING SEQUENCE clause 244
Definition 71
FILE-CONTROL paragraph 236
OBJECT-COMPUTER paragraph 216
Scope of 107
SPECIAL-NAMES paragraph 220
Alphabets 65
Alphanumeric category 120, 319, 344
Alphanumeric character 8
Alphanumeric character set 61, 793
Alphanumeric class 120
Alphanumeric coded character set 61, 793
Alphanumeric data item 120
Alphanumeric functions 610
Alphanumeric literals 76
Continuation of 26, 27
ALPHANUMERIC phrase
CODE-SET clause 298
COLLATING SEQUENCE clause 244
INITIALIZE statement 493
MERGE statement 508
SORT statement 568
SPECIAL-NAMES paragraph 219, 220
Alphanumeric-edited category 120, 344
Alphanumeric-edited data item 120
ALPHANUMERIC-EDITED phrase
INITIALIZE statement 493
ALSO phrase
EVALUATE statement 476
ALTERNATE RECORD KEY clause 243, 609
FILE-CONTROL paragraph 236
AND operator
In combined conditions 153
In complex conditions 153
AND phrase
SEARCH statement 552
ANNUITY function 628
ANY LENGTH clause 97, 286, 442, 456
ANY LENGTH STRUCTURE clause 220
ANY phrase
EVALUATE statement 476
Any-length elementary item 117
Any-length elementary items 760
Any-length-structure-name 68, 71
Apostrophe 63
Archaic language element flagging 5
Archaic language elements 5, 873
Continuation of COBOL words 873
Identifier-n (text-n) in a COPY statement 873
MOVE of nonnumeric figurative constants to numeric items 873
NEXT SENTENCE phrase 491, 552, 873
ON OVERFLOW phrase of CALL 873
Arguments 20
Arguments, intrinsic functions 610
Arithmetic 128, 131
Native 131
Standard 131
Standard-binary 136
Standard-decimal 138
ARITHMETIC clause 132, 207
Arithmetic compute 465
Arithmetic expressions 21, 131
COMPUTE statement 465
EVALUATE statement 476
Parenthesis in 131
Sign condition 152
Arithmetic Operators 128
Arithmetic statements 435
ADD statement 435, 449
COMPUTE statement 435, 465
Data conversion 435
decimal point alignment 435
DIVIDE statement 435, 473
maximum operand size 435
MULTIPLY statement 435, 521
SUBTRACT statement 435, 583
AS phrase
CALL statement 455
CLASS-ID paragraph 200
EXTERNAL clause 259, 260, 311
FUNCTION-ID paragraph 202
INTERFACE-ID paragraph 203
PROGRAM-ID paragraph 211
REPOSITORY paragraph 230
ASCENDING KEY phrase
MERGE statement 508
OCCURS clause 333
SEARCH statement 555
SORT statement 568
ASIN function 629
ASSIGN clause
FILE-CONTROL paragraph 236
Asterisk (*) comment line 27
At end condition 539, 540, 545
Definition 432
AT END phrase
READ statement 535
RETURN statement 545
SEARCH statement 552
AT END-OF-PAGE phrase

WRITE statement 602
AT EOP phrase
WRITE statement 602
ATAN function 630
ATTRIBUTE phrase
SET statement, attribute setting 558
AUTO clause 288
Automatic data 125, 417
Automatic items 125
AUTOMATIC phrase
LOCK MODE clause 247
AWAY-FROM-ZERO phrase
DEFAULT ROUNDED clause 208
ROUNDED phrase 432

B

B PICTURE symbol 341
BACKGROUND-COLOR clause 289
B-AND operator 128
BASE class 715
BASED clause 126, 278, 290
Based data 126
Based data item 404
ALLOCATE statement 453
SET statement 561
Based entry 126
Basic calendar date format
Definition 612
Basic combined date and time format 616
Basic common time format
Definition 614
Basic common time format with integer seconds repre-
sentation
Definition
Basic common time format
Definition 614
Basic common time3 format with fractional seconds
Definition 614
Basic letters 63
Basic local time format 615
Basic offset time format 615
Basic ordinal date format
Definition 613
Basic UTC time format 615
Basic week date format
Definition 613
BEFORE ADVANCING phrase
WRITE statement 602
BEFORE phrase
INSPECT statement 498
PERFORM statement 528
BEFORE REPORTING phrase
USE statement 593
BELL clause 291
BELL phrase
SET statement, attribute setting 558
BINARY phrase
USAGE clause 391
BINARY-CHAR phrase
USAGE clause 391
BINARY-DOUBLE phrase
USAGE clause 391
BINARY-LONG phrase
USAGE clause 391
BINARY-SHORT phrase
USAGE clause 391
BIT 319
Bit data item 319, 393
Bit group 791
Bit group item 319
BIT phrase
USAGE clause 391
BLANK clause 292
Blank line 26, 27
BLANK WHEN ZERO clause 293, 342, 404
BLINK clause 294
BLINK phrase
SET statement, attribute setting 558
BLOCK CONTAINS clause 295
File description entry 259
Boolean category 120, 121, 319, 344
Boolean character 8
Boolean class 120
BOOLEAN clause 149
Boolean compute 465
Boolean condition 148
Boolean data item 121
Boolean expressions 139
COMPUTE statement 465
EVALUATE statement 476
Parenthesis in 140
Relation condition 144
Boolean functions 610
Boolean literals 78
Continuation of 26, 27
Boolean operators 128
BOOLEAN phrase
CLASS clause 296
INITIALIZE statement 493
BOOLEAN-OF-INTEGER function 631
B-OR operator 128
Braces 19
Brackets 19
B-XOR operator 128
BY CONTENT phrase
CALL statement 454
INVOKE statement 504
Procedure division header 94
BY phrase
COPY statement 33
DIVIDE statement 473
INITIALIZE statement 493
INSPECT statement 499
MULTIPLY statement 521

PERFORM statement 529
SET statement, pointer arithmetic 558
VARYING clause 407
BY REFERENCE phrase
CALL statement 454
INVOKE statement 504
Procedure division header 94, 410
BY VALUE phrase
CALL statement 455
INVOKE statement 504
Procedure division header 95, 410
BYTE-LENGTH function 632
BYTE-LENGTH phrase
Constant entry 270

C

Calendar date format 612
CALL statement 454
CALL-CONVENTION directive 44, 859
Call-convention-name 73, 74
call-convention-name 44
CANCEL statement 417, 460, 484
CAPACITY 333
Case mapping 64
Category of data
Alphabetic 120, 344
Alphanumeric 120, 319, 344
Alphanumeric-edited 120, 344
Boolean 120, 121, 319, 344
Boolean literal 78
Data-pointer 120, 121
Function-pointer 120, 121
Index 120, 121, 393
National 120, 121, 319, 344
National literal 79
National-edited 120, 121, 344
Numeric 120, 122, 344
Numeric literal 78
Numeric-edited 122, 344
Object-reference 120, 122, 395
Program-pointer 120, 122
CF phrase
TYPE clause 384
CH phrase
TYPE clause 384
CHAR function 633
Character
Alphabetic 8
Alphanumeric 8
Boolean 8
National 14
Numeric 14
CHARACTER CLASSIFICATION clause 217
OBJECT-COMPUTER paragraph 216
Character sets 61, 219, 793
Character substitution 65
CHARACTERS phrase
ALLOCATE statement 452
BLOCK CONTAINS clause 295
INSPECT statement 498
RECORD clause 361
Character-string 9, 68
CHAR-NATIONAL function 634
Choice indicators 19
Class
BASE 715
Parameterized 188
Class and category of data 119
CLASS clause 149, 296
SPECIAL-NAMES paragraph 219
Class condition 148
BOOLEAN phrase 149
Class inheritance 188
Class of data
Alphanumeric 120
Boolean 120
Index 120
National 120
Numeric 120
Object 120
Pointer 120
CLASS phrase
USAGE clause 391
Class polymorphism 180
Class-definition 195
CLASS-ID paragraph 200
Class-name 219
CLASS clause 296
CLASS-ID paragraph 200
Object orientation 68, 71
Scope of 107, 109
SPECIAL-NAMES paragraph 219
Truth value 68, 71
Clause 9
CLOSE statement 461, 462
Implied 511, 512, 572, 573
Closing delimiter 84
COBOL character repertoire 62
COBOL compilation group 193
COBOL library 30
COBOL reserved words 74
COBOL source program structure 191
COBOL word 68
CODE clause 297
Report description entry 273
Coded character set 61, 793
Code-name 74
SPECIAL-NAMES paragraph 220
CODE-SET clause 61, 298, 373
File description entry 259
Collating sequence 219
COLLATING SEQUENCE clause 244
FILE-CONTROL paragraph 236
OBJECT-COMPUTER paragraph 216

COLLATING SEQUENCE phrase
MERGE statement 508
SORT statement 568
Collating sequences 65
Colon 63
Separator 85
Color number 179
COLUMN (COL) phrase
ACCEPT statement 443
DISPLAY statement 470
COLUMN clause 300
Combined conditions 153
Combined date and time format 616
COMBINED-DATETIME 635
Comma 20
Interchangeable with semicolon 84
Separator 84
Comment 26, 27
Inline 27, 28
Line 27, 28
Comment indicator 23, 24
COMMON clause
PROGRAM-ID paragraph 211
Common exception processing 803
Common phrases 432
Common program 126
Scope of 109
Common time format 614
COMP phrase
USAGE clause 391
Comparison
Between object reference identifiers 148
Locale-based 146, 147
Of alphanumeric operands 145
Of boolean operands 146
Of data-pointer operands 148
Of index data items 147
Of mixed operands 145
Of national operands 146
Of numeric operands 145
Of strongly-typed group items 147
Compatibility
FLAG-85 directive 49, 50
Compatibility of Variable-length groups
Definition 118
Compilation group 191, 770
Compilation stage 30
Compilation unit 6, 191, 771
Compilation variable name 45
Compilation-variable-name 45, 68, 71, 110
Compiler directing statement 31
COPY statement 31
REPLACE statement 31
Compiler directive 40
Compiler directive indicator 24
Compiler-directive word 40, 68, 163
Compile-time arithmetic expressions 41, 270
Compile-time boolean expressions 41
Complex conditions 153
AND operator 153
NOT operator 153
OR operator 153
Composite of operands 132, 435, 813
ADD statement 449
DIVIDE statement 474
MULTIPLY statement 521
SUBTRACT statement 583
COMPUTATIONAL phrase
USAGE clause 391
COMPUTE statement 435, 465
Computer's character set 61, 793
Computer's coded character set 61, 111, 793
Computer-name 74
OBJECT-COMPUTER paragraph 216
SOURCE-COMPUTER paragraph 215
Concatenation expressions 142
Concatenation operator 128
Concepts 751
CONDITION 593
Condition 142
Abbreviated combined relation condition 154
Boolean condition 148
Class condition 148
Complex 153
Condition-name condition 151
EVALUATE statement 476
Evaluation order rules 153
IF statement 491
Omitted argument condition 152
Sign condition 152
Switch-status condition 151
Conditional compilation 53, 858
Conditional expression 142, 529, 846
EVALUATE statement 476
Conditional phrase 413, 415
Conditional statement 413
Conditional variable 151
Definition 106
Conditionally-processed compilation group 31
Condition-name 68, 71, 106, 151
Qualified with subscripts 106
Scope of 107, 108
Subscripted 89
Switch status 106
Condition-name condition 151
Configuration section 214
Conformance 185
Conformance for object orientation 185
Conforming implementation 6, 7
Conforming run unit 6
Constant conditional expression 42
Constant entry 270
CONSTANT RECORD 370
CONSTANT RECORD clause 304
Constant-name 68, 71, 107, 270
Scope of 108

Contained source unit 192
Contained statement 416
CONTENT phrase
INVOKE statement 504
SET statement 559
VALIDATE-STATUS clause 398
Content validation 598
Context-sensitive word 75
Context-sensitive word list 159
Contiguity of data items 118
Continuation indicator 23
Continuation of lines 26, 27
CONTINUE statement 419, 467, 483, 492, 553
Continued line 27
Control break 305, 318
Control break processing 486
CONTROL clause 305, 486, 596
Report description entry 273
CONTROL FOOTING phrase
TYPE clause 384
CONTROL HEADING phrase
TYPE clause 384
CONVERTING phrase
INSPECT statement 498
COPY statement 31, 33, 47, 53
Nesting 36
Correspondence
Variable-length groups 119
CORRESPONDING (CORR) phrase 435
ADD statement 435, 449
FILLER clause 435
MOVE statement 435, 513
OCCURS clause 435
REDEFINES clause 435
RENAMES clause 435
SUBTRACT statement 435, 583
COS function 636
COUNT IN phrase
UNSTRING statement 589
CR PICTURE symbol 341
CRT status 177
CRT STATUS clause
SPECIAL-NAMES paragraph 219
Cultural adaptability 798
Cultural elements 66
Currency sign 219
CURRENCY SIGN clause
SPECIAL-NAMES paragraph 219
Currency string 223
Currency symbol 219, 223
currency_symbol 67
Current screen item 179
Current volume pointer 168, 754
CURRENT-DATE function 637
Cursor 178
CURSOR clause
SPECIAL-NAMES paragraph 219
Cursor locator 178
CYCLE phrase
EXIT statement 481

D

d_fmt 67
DATA BY phrase
INITIALIZE statement 493
Data conversion 435, 446
ACCEPT statement 446
MOVE statement 515
Data description entry 263, 276
Data division 257
Data division entries 257
Data item 90
Data-address-identifier 102
Data-name 309
Data description entry 276
Definition 71
Identifier 91
Qualified 91
Scope of 108
Subscripted 89, 91
Data-pointer 66, 452, 485
Definition 395
Data-pointer category 103, 120, 121
Data-pointer data item 121
DATA-POINTER phrase
INITIALIZE statement 493
Date format 612
Date forms 619
DATE phrase
ACCEPT statement 443
DATE-OF-INTEGER function 638
DATE-TO-YYYYMMDD function 639
DAY phrase
ACCEPT statement 443
Day subfield 612
DAY-OF-INTEGER function 640
DAY-OF-WEEK phrase
ACCEPT statement 443
Day-of-week subfield 613
Day-of-year subfield 613
DAY-TO-YYYYDDD function 641
DB PICTURE symbol 341
DE phrase
TYPE clause 384
Decimal point 219
Alignment 421
Decimal point alignment 435
DECIMAL POINT IS COMMA clause
SPECIAL-NAMES paragraph 219
Decimal128 format 138
DECIMAL-POINT IS COMMA clause 682, 684, 685
Declarative statement 413
USE statement 593
DECLARATIVES 409, 412

Declaratives 409, 412
Normal completion 424
De-editing 50, 515, 517
DEFAULT clause 307
DEFAULT phrase
INITIALIZE statement 493
SET statement, locale 558
DEFAULT ROUNDED clause 208
DEFINE directive 45
Defined condition 42
Definitions 8
DELETE statement 468
DELIMITED BY phrase
STRING statement 580
UNSTRING statement 589
DELIMITED phrase
ANY LENGTH STRUCTURE clause 220
Delimited scope statement 415
DELIMITER IN phrase
UNSTRING statement 589
DEPENDING ON phrase
GO TO statement 488
OCCURS clause 333
RECORD clause 361
DESCENDING KEY phrase
MERGE statement 508
OCCURS clause 333
SEARCH statement 555
SORT statement 568
DESTINATION clause 308
DETAIL phrase
TYPE clause 384
Device-name 74
FILE-CONTROL paragraph 236
SPECIAL-NAMES paragraph 219
DISPLAY phrase
USAGE clause 391
DISPLAY screen statement 470, 565
DISPLAY statement 221, 470
DISPLAY-OF function 642
DIVIDE statement 435, 473
Division 135
Division header 191
DOWN phrase
SET statement, index arithmetic 557, 559
SET statement, pointer arithmetic 558
DUPLICATES phrase
ALTERNATE RECORD KEY clause 243
SORT statement 568
DYNAMIC 333
Dynamic access mode 168, 753
DYNAMIC phrase
ACCESS MODE clause 236, 242
Dynamic storage 452, 485
Dynamic-capacity tables 766
Definition 115
Dynamic-capacity-table
Format 333

E

E function 643
E PICTURE symbol 341
EC 593
EC-ALL exception 59, 423, 426, 804
EC-ARGUMENT exception 423, 426, 804
EC-ARGUMENT-FUNCTION exception 426, 611, 722
EC-ARGUMENT-IMP exception 426
EC-BOUND exception 423, 426, 804
EC-BOUND-IMP exception 426
EC-BOUND-ODO exception 337, 426
EC-BOUND-OVERFLOW exception 117, 426
EC-BOUND-PTR exception 290, 426
EC-BOUND-REF-MOD exception 97, 426
EC-BOUND-SET exception 426, 567
EC-BOUND-SUBSCRIPT exception 90, 426
EC-BOUND-SUBSCRIPT exception condition 563, 564
EC-BOUND-TABLE-LIMIT exception 117, 426, 567
EC-DATA exception 423, 426, 804
EC-DATA-CONVERSION exception 426, 515, 642, 681
EC-DATA-IMP exception 426
EC-DATA-INCOMPATIBLE exception 136, 138, 426, 430, 431, 435, 446, 447, 488, 517, 518, 599, 732
EC-DATA-INTEGRITY exception 426
EC-DATA-NOT-DECIMAL-ENCODING exception 431, 517
EC-DATA-NOT-FINITE exception 431
EC-DATA-PTR-NULL exception 290, 426, 565
EC-FLOW exception 423, 426, 804
EC-FLOW-GLOBAL-EXIT exception 426, 483
EC-FLOW-GLOBAL-GOBACK exception 426, 490
EC-FLOW-IMP exception 427
EC-FLOW-RELEASE exception 427, 542
EC-FLOW-REPORT exception 427, 596
EC-FLOW-RETURN exception 427, 545
EC-FLOW-SEARCH exception 427, 567
EC-FLOW-USE exception 427, 594
EC-FUNCTION exception 427
EC-FUNCTION-NOT-FOUND exception 427
EC-FUNCTION-PTR-INVALID exception 427, 565
EC-FUNCTION-PTR-NULL exception 95, 427
EC-IMP exception 423, 427, 482, 804
EC-IMP- exception 423
EC-I-O exception 174, 423, 427, 527, 644, 645, 804
EC-I-O-AT-END exception 169, 427, 755
EC-I-O-EOP exception 427, 607
EC-I-O-EOP-OVERFLOW exception 427, 607
EC-I-O-FILE-SHARING exception 169, 427
EC-I-O-IMP exception 169, 427
EC-I-O-INVALID-KEY exception 169, 174, 427
EC-I-O-LINAGE exception 325, 427
EC-I-O-LOGIC-ERROR exception 169, 363, 427
EC-I-O-PERMANENT-ERROR exception 169, 427
EC-I-O-RECORD-OPERATION exception 169, 427
EC-LOCALE exception 423, 427
EC-LOCALE-IMP exception 427
EC-LOCALE-INCOMPATIBLE exception 146, 147, 427

EC-LOCALE-INVALID exception 67, 427
EC-LOCALE-INVALID-PTR exception 427, 566
EC-LOCALE-MISSING exception 67, 427, 566, 667, 668, 669, 670, 684
EC-LOCALE-SIZE exception 352, 427
EC-OO exception 423, 427, 804
EC-OO-CONFORMANCE exception 98, 99, 427, 438
EC-OO-EXCEPTION exception 425, 428
EC-OO-IMP exception 428
EC-OO-METHOD exception 182, 428, 507
EC-OO-NULL exception 428, 506
EC-OO-RESOURCE exception 428, 715
EC-OO-UNIVERSAL exception 428, 438, 507
EC-ORDER exception 423, 428
EC-ORDER-IMP exception 428
EC-ORDER-NOT-SUPPORTED exception 428, 700
EC-ORDER-NOT-SUPPORTED exception condition 737
EC-OVERFLOW exception 423, 428, 804
EC-OVERFLOW-IMP exception 428
EC-OVERFLOW-STRING exception 428, 581
EC-OVERFLOW-UNSTRING exception 428, 591
EC-PROGRAM exception 423, 428, 458, 804
EC-PROGRAM-ARG-MISMATCH exception 428, 438, 457
EC-PROGRAM-ARG-OMITTED exception 95, 428, 459, 507
EC-PROGRAM-CANCEL-ACTIVE exception 428, 460
EC-PROGRAM-IMP exception 428
EC-PROGRAM-NOT-FOUND exception 95, 104, 428, 457
EC-PROGRAM-PTR-NULL exception 428, 457
EC-PROGRAM-RECURSIVE-CALL exception 428, 458
EC-PROGRAM-RESOURCES exception 95, 428, 457
EC-RAISING exception 428
EC-RAISING-IMP exception 428
EC-RAISING-NOT-SPECIFIED exception 428, 483
EC-RANGE exception 423, 428, 804
EC-RANGE-IMP exception 428
EC-RANGE-INDEX exception 336, 428
EC-RANGE-INDEX exception condition 563, 564
EC-RANGE-INSPECT-SIZE exception 428, 502
EC-RANGE-INVALID exception 428, 437
EC-RANGE-PERFORM-VARYING exception 429, 530
EC-RANGE-PTR exception 429, 566
EC-RANGE-SEARCH-INDEX exception 429, 554, 555
EC-RANGE-SEARCH-NO-MATCH exception 429, 554, 555, 556
EC-REPORT exception 423, 429, 804
EC-REPORT-ACTIVE exception 429, 497
EC-REPORT-COLUMN-OVERLAP exception 301, 429
EC-REPORT-FILE-MODE exception 429, 497
EC-REPORT-IMP exception 429
EC-REPORT-INACTIVE exception 429, 487, 587
EC-REPORT-LINE-OVERLAP exception 327, 429
EC-REPORT-NOT-TERMINATED exception 429, 464
EC-REPORT-PAGE-LIMIT exception 327, 429
EC-REPORT-PAGE-WIDTH exception 301, 429
EC-REPORT-SUM-SIZE exception 378, 429
EC-REPORT-VARYING exception 407, 429
EC-SCREEN exception 429, 447
EC-SCREEN-FIELD-OVERLAP exception 429, 447, 471, 472
EC-SCREEN-IMP exception 429
EC-SCREEN-ITEM-TRUNCATED exception 303, 429, 472
EC-SCREEN-LINE-NUMBER exception 329, 429, 472
EC-SCREEN-STARTING-COLUMN exception 303, 429, 472
EC-SIZE exception 423, 429, 433, 435, 804
EC-SIZE-ADDRESS exception 429, 565
EC-SIZE-EXPONENTIATION exception 132, 136, 137, 139, 429, 434
EC-SIZE-IMP exception 429
EC-SIZE-OVERFLOW exception 133, 134, 136, 138, 429, 434
EC-SIZE-TRUNCATION exception 134, 209, 210, 429, 434
EC-SIZE-UNDERFLOW exception 133, 136, 138, 429, 434, 809
EC-SIZE-ZERO-DIVIDE exception 429, 434
EC-SORT-MERGE exception 423, 429, 804
EC-SORT-MERGE-ACTIVE exception 429, 511, 572, 573
EC-SORT-MERGE-FILE-OPEN exception 430, 510, 511, 571
EC-SORT-MERGE-IMP exception 430
EC-SORT-MERGE-RELEASE exception 363, 430, 511, 572
EC-SORT-MERGE-RETURN exception 430, 545
EC-SORT-MERGE-SEQUENCE exception 430, 510
EC-STORAGE exception 423, 430
EC-STORAGE-IMP exception 430
EC-STORAGE-NOT-ALLOC exception 430, 485
EC-STORAGE-NOT-AVAIL exception 430, 453
EC-USER exception 410, 423, 430, 482, 483, 804
EC-VALIDATE exception 430, 600
EC-VALIDATE-CONTENT exception 430, 600
EC-VALIDATE-FORMAT exception 430, 600
EC-VALIDATE-IMP exception 430
EC-VALIDATE-RELATION exception 430, 600
EC-VALIDATE-VARYING exception 408, 430
Editing rules 349
Elementary items 111
Ellipses 19, 21
ELSE NEXT SENTENCE phrase
    IF statement 491
ELSE phrase
    IF statement 491
END CLASS marker 198
END DECLARATIVES 409, 412
END FACTORY marker 198
END FUNCTION marker 198
END INTERFACE marker 198
END MARKERS 198
END METHOD marker 198
END OBJECT marker 198
End of COBOL source program 198

END PROGRAM marker 198
END-ACCEPT phrase 443
END-ADD phrase 449
END-CALL phrase 454
END-COMPUTE phrase 465
END-DELETE phrase 468
END-DISPLAY phrase 470
END-DIVIDE phrase 473
END-EVALUATE phrase 476
END-IF phrase
IF statement 491
END-MULTIPLY phrase 521
END-OF-PAGE phrase
WRITE statement 602
END-READ phrase 535
END-RETURN phrase 545
END-REWRITE phrase 547
END-SEARCH phrase 552
END-START phrase 575
END-STRING phrase 580
END-SUBTRACT phrase 583
END-UNSTRING phrase 589
END-WRITE phrase 602
ENTRY-CONVENTION clause 208, 859
Entry-convention-name 74, 208
Entry-name clause 309
Environment division 213
EO 593
EOP phrase
WRITE statement 602
EQUAL phrase
SEARCH statement 552
Equivalent arithmetic expression 618
ERASE clause 310
ERROR clause 398
Error indication 598
ERROR phrase
STOP statement 579
USE statement 593
VALIDATE-STATUS clause 398
ERROR PROCEDURE phrase
USE statement 593
EVALUATE directive 46
EVALUATE statement 476
Evaluation of conditional expressions 153
Evaluation of conditions 846
Evaluation rules for conditions 155
EXCEPTION CONDITION 593
Exception condition handling 422
Exception conditions 422, 426
Fatal 424
Non-fatal 424
Exception declaratives 755
Exception functions 755
Exception handling 422, 593, 755
EXCEPTION OBJECT 593
Exception object 425, 593
EXCEPTION phrase 489
ACCEPT statement 443
DISPLAY statement 470
EXIT statement 481
RAISE statement 534
USE statement 593
EXCEPTION PROCEDURE phrase
USE statement 593
Exception status indicators 422
EXCEPTION-FILE function 644
EXCEPTION-FILE-N function 645
EXCEPTION-LOCATION function 646
EXCEPTION-LOCATION-N function 647
Exception-names 76, 423, 426
EXCEPTION-OBJECT 99, 423
RAISE statement 534
EXCEPTION-STATEMENT function 648
EXCEPTION-STATUS function 423, 649
Execution 416
EXIT FUNCTION statement 11, 481, 484, 532
EXIT METHOD statement 11, 481, 484, 532
EXIT PARAGRAPH statement 11, 481, 484
EXIT PERFORM statement 11, 481, 484
EXIT PROGRAM statement 11, 481, 483, 489, 532
EXIT SECTION statement 11, 481, 484
EXIT statement 481
EXP function 650
EXP10 function 651
Expanded compilation group 31
EXPANDS phrase
REPOSITORY paragraph 189
Expected capacity
Exceeding 117
Expected capacity of a dynamic table
Definition 115
Explicit attributes 257
Explicit reference 106
Explicit scope terminators 415
Explicit transfer of control 418
Exponent 78
Exponentiation 132, 135
Expression
Arithmetic 21, 131
Boolean 139, 144, 465, 476
Compile-time arithmetic 41
Compile-time boolean 41
Concatenation 142
Conditional 142, 846
Constant conditional 42
EXTEND phrase
OPEN statement 523
USE statement 593
Extended calendar date format
Definition 612
Extended combined date and time forma 616
Extended common time format
Definition 614
Extended common time format with fractional seconds
Definition 614

Extended common time format with integer seconds representation
Definition 614
Extended letters 63, 743
Extended local time format 615
Extended offset time format 615
Extended ordinal date format
Definition 613
Extended relational operator 129
Extended UTC time format 615
Extended week date format
Definition 613
Extension language elements 4
Extensions
Nonstandard 4
Standard 4
External 124
EXTERNAL clause 311, 370
Data description entry 276
File description entry 261
External data items 404
External items 124
EXTERNAL phrase
File description entry 259
External repository 164
Externalized names 70
External-locale-name 74
Externally provided functionality 5

F

FACTORIAL function 652
FACTORY clause
METHOD-ID paragraph 204
Factory data 188, 818
Factory definition 201
Factory method 180, 818
Factory object 180, 201, 818
Life cycle for 189
FACTORY paragraph 201
FACTORY phrase 439
USAGE clause 391
Factory-definition 195
FactoryObject method 715
FALSE phrase
EVALUATE statement 476
SET statement, condition setting 557
Fatal exception conditions 424
FD entry 259
Feature-name 74
FILE-CONTROL paragraph 238
RECORD DELIMITER clause 250
SPECIAL-NAMES paragraph 219
Figurative constant 81
INITIALIZE statement 495
Symbolic characters in 84
File
logical 165
Physical 165
Physical aspects 111
File attribute conflict condition 525
File attributes 166
File connector 165
OPEN statement 523
File control entry 236
File description entry 259
File operations 754
File organization 166, 751
FILE phrase
REWRITE statement 547
WRITE statement 602
File position indicator 168
File processing 752
File section 258
File sharing 174, 254, 756
File sharing conflict condition 170, 437
FILE STATUS clause 246
FILE-CONTROL paragraph 236
FILE-CONTROL paragraph 236
File-name 68, 71
Scope of 108
Files 165, 751
Indexed 167
Relative 167
Sequential 167
FILLER clause 309, 382
FILLER phrase
INITIALIZE statement 493
FINAL phrase
CLASS-ID paragraph 200
SUM clause 377
TYPE clause 384
FIRST DETAIL phrase
PAGE clause 339
FIRST phrase
INSPECT statement 499
START statement 575
Fixed file attributes 166, 525
Fixed indicators 23
Fixed insertion editing 350
Fixed-capacity table
Comparing to dynamic-capacity table 116
Fixed-capacity tables
Definition 114
Fixed-length group
Definition 118
Fixed-length records 752
Fixed-point numeric literal 78
FLAG-85 directive 49, 50
Flagging and warning mechanisms
Archaic language elements 5
FLAG-85 directive 49, 50
FLAG-NATIVE-ARITHMETIC directive 52
Nonstandard extensions 5
Obsolete language elements 5

Processor-dependent elements 4
Prototype versus repository 164
Syntax violations 3
FLAG-NATIVE-ARITHMETIC directive 52, 132
FLOAT-BINARY-16
USAGE clause
FLOAT-BINARY-34
USAGE clause 391
FLOAT-BINARY-16 phrase
USAGE clause 394
FLOAT-BINARY-34 phrase
USAGE clause 136, 394
FLOAT-BINARY-7
USAGE clause 391
FLOAT-BINARY-7 phrase
USAGE clause 394
FLOAT-DECIMAL-16
USAGE clause
FLOAT-DECIMAL-34
USAGE clause 391
FLOAT-DECIMAL-16 phrase
USAGE clause 394
FLOAT-DECIMAL-34 phrase
USAGE clause 138, 394
FLOAT-EXTENDED phrase
USAGE clause 391
Floating indicators 24
Floating-point numeric literal 78
FLOAT-LONG phrase
USAGE clause 391
FLOAT-SHORT phrase
USAGE clause 391
FOOTING phrase
PAGE clause 339
FOR phrase
INSPECT statement 498
VALIDATE-STATUS clause 398
FOR REMOVAL phrase
CLOSE statement 462
FOREGROUND-COLOR clause 312
FOREVER phrase
DELETE statement 468
OPEN statement 523
READ statement 535
RETRY phrase 437
REWRITE statement 547
WRITE statement 602
Formal parameter 411, 438
FORMAT clause 313, 373
File description entry 259
FORMAT phrase
VALIDATE-STATUS clause 398
Format validation 598
PICTURE clause 346
FORMATTED-CURRENT-DATE function 653
FORMATTED-DATE function 654
FORMATTED-DATETIME function 655
FORMATTED-TIME function 656
Forms of arithmetic
Concepts 807
frac_digits 67
FRACTION-PART function 657
FREE statement 485
Free-form reference format 27
FROM clause 315
FROM phrase 443
ACCEPT statement 443
Constant entry 270
PERFORM statement 529
RELEASE statement 542
REWRITE statement 547
SUBTRACT statement 583
VARYING clause 407
WRITE statement 602
FULL clause 316
Function keys 177
FUNCTION phrase
EXIT statement 11, 481
Function summary 619
Function-definition 195
FUNCTION-ID paragraph 202
Function-identifier 92
Function-pointer
Definition 396
Function-pointer category 120, 121
Function-pointer data item 121
FUNCTION-POINTER phrase
USAGE clause 391
Function-prototype 193
Function-prototype-name 68, 71
Functions
Intrinsic 162, 163, 610
User-defined 189, 230

G

General formats 18
General rules 20
GENERATE statement 386, 486, 587, 594
GIVING phrase
ADD statement 449
DIVIDE statement 473
MERGE statement 508
MULTIPLY statement 521
SORT statement 568
SUBTRACT statement 583
GLOBAL clause 317, 370, 389
Constant entry 270
File description entry 259
Report description entry 273
Global names 107, 124
GLOBAL phrase
USE statement 593
GO TO statement 488
GOBACK statement 57, 489, 532

Group
Strongly-typed 122
Group items 111
Group move 515
GROUP-USAGE 319
GROUP-USAGE clause 319, 384, 598

H

HEADING phrase
PAGE clause 339
HIGHEST-ALGEBRAIC function 658
HIGHLIGHT clause 320
HIGHLIGHT phrase
SET statement, attribute setting 558
HIGH-VALUE/HIGH-VALUES figurative constant 83
Hours subfield 614

I

Identification division 199
Identifier
Data-name 91
Identifiers 90
IEEE 754-2008 810
IEEE Std 754 2008 2
IEEE STD 754-2008 138
IEEE Std 754-2008 136
IF directive 53
IF statement 491
IGNORING LOCK phrase
READ statement 535
IMP directive 40, 41
Imperative statement 413
Implementation
Nonstandard extensions 4
Reserved words 4
Standard extensions 4
Implementor-defined
Intermediate data item 133
Language element list 717
Language elements 3
Native arithmetic 132
Record types 752
IMPLEMENTS clause 201, 206
Implicit attributes 257
Implicit reference 106
Implicit scope terminators 415
Implicit transfer of control 418
IN phrase
COPY statement 33
With identifiers 90
With qualification 88
Incompatibility flagging 49, 50
Incompatible data 430
Index
Definition 336
Index-name 89
Index category 120, 121, 393
Index class 120
Index data item 121, 392
Index functions 610
INDEX phrase
USAGE clause 391
INDEXED BY phrase 89, 552, 562
OCCURS clause 333
Indexed files 167
Indexed organization 752
INDEXED phrase
ORGANIZATION clause 249
Index-name 68, 72
Scope of 108, 109
Indicator area 25
Indicators 23
Comment 23, 24
Compiler directive 24
Continuation 23
Fixed 23
Floating 24
Literal continuation 24
Source 24
INFINITY phrase
CLASS clause 149
Inheritance
Conforming 188
INHERITS clause
CLASS-ID paragraph 200
INTERFACE-ID paragraph 203
INITIAL clause
PROGRAM-ID paragraph 211
Initial data 125, 263, 417
Initial items 125, 417
Initial program 126
Initial state 417
CANCEL statement 417, 460
INITIALIZE statement 400, 404, 493
INITIALIZED 333
INITIALIZED phrase
ALLOCATE statement 452
INITIATE statement 497, 525, 594
Inline comment 27, 28
Inline method invocation 97
Input distribution 598
INPUT phrase
OPEN statement 523
USE statement 593
INPUT PROCEDURE phrase
SORT statement 568
Input-output section 235
Insertion character 350
INSPECT statement 498
INSPECT statement examples 850
Instance data 188
Instance method 817
Instance object 180, 206, 817

Life cycle for 189
instance-definition 195
int_curr_symbol 67
int_frac_digits 67
Integer
In general formats and rules 21
Integer date form 619
INTEGER function 659
Integer functions 610
Integer literal 78
Integer-n 19
INTEGER-OF-BOOLEAN function 660
INTEGER-OF-DATE function 661
INTEGER-OF-DAY function 662
INTEGER-OF-FORMATTED-DATE function 663
INTEGER-PART function 664
Interface
Parameterized 189
Interface-definition 196
INTERFACE-ID paragraph 203
Interface-name 68, 72
Scope of 109
Interfaces 185
Internal items 124
INTO phrase
DIVIDE statement 473
READ statement 535
RETURN statement 545
STRING statement 580
UNSTRING statement 589
Intrinsic function summary 619
Intrinsic functions 610
Intrinsic-function-name 76
INVALID clause 321
Invalid key condition 173, 577
Definition 432
INVALID KEY phrase
DELETE statement 468
READ statement 535
REWRITE statement 547
START statement 575
WRITE statement 602
INVALID phrase
VALUE clause 400
Invocation operator 128
INVOKE statement 504
I-O phrase
OPEN statement 523
USE statement 593
I-O status 169, 755
I-O-CONTROL paragraph 255
ISO 1001 2
ISO 1001:1986 250
ISO 1989:1985 50, 860
ISO 1989:1985/Amd 1 860
ISO 1989:1985/Amd 1:1992 50
ISO 1989:1985/Amd 2:1994 50
ISO 8601 2, 612, 614, 862
ISO 8601:2004 619
ISO/IEC 10646 2, 10, 12, 16, 61, 63, 64, 112, 226, 747, 793, 867
ISO/IEC 14651 2, 72, 229, 428, 700, 725, 737, 797
ISO/IEC 1989:2002 49, 50, 860, 861, 862
ISO/IEC 646 2, 61, 63, 77, 170, 226, 793, 795, 797
ISO/IEC 9945 2, 66, 67
ISO/IEC TR 10176 867
ISO/IEC TR 10176:2003 743
Item identification 419

J

Julian date form 619
JUSTIFIED clause 322, 401, 580
Data description entry 276

K

Key of reference 577
KEY phrase
READ statement 535
START statement 575
Keywords 18
Known errors in the standard 875

L

LAST DETAIL phrase
PAGE clause 339
LAST EXCEPTION phrase 489
EXIT statement 481
Last exception status 804
Definition 423
LAST phrase
START statement 575
Last-used state 417, 418
LC_ALL 66
SET statement, locale 558
LC_COLLATE 66, 83, 420
SET statement, locale 558
LC_CTYPE 66, 420, 673, 711
SET statement, locale 558
LC_MESSAGES 66
SET statement, locale 558
LC_MONETARY 66, 420
SET statement, locale 558
LC_NUMERIC 66
SET statement, locale 558
LC_TIME 66, 420
SET statement, locale 558
LEADING phrase
INSPECT statement 498
SIGN clause 374
LEAP-SECOND directive 54
LEFT phrase

SYNCHRONIZED clause 381
LENGTH function 665
LENGTH phrase
Constant entry 270
START statement 575
Letter mapping 747
Level-number 69, 72, 111, 323
Data description entry 276
Screen description entry 281
Levels 111
Lexical elements 68
Library text 32
Library-name 33, 74
Life cycle
Data 124
Object 189
LINAGE clause 324
File description entry 259
Linage concepts 754
LINAGE-COUNTER 104, 109
Qualified 88
Line
Comment 27, 28
Continuation 27
Floating indicator 24
Slant (/) comment line 27
Source text 23
LINE (LINES) phrase
WRITE statement 602
LINE clause 326
LINE NUMBER phrase
ACCEPT statement 443
DISPLAY statement 470
LINE-COUNTER 105, 109, 273, 327
Qualified 88
LINES AT BOTTOM phrase
LINAGE clause 324
LINES AT clause
File description entry 259
LINES AT TOP phrase
LINAGE clause 324
Linkage section 265
LISTING directive 55
Literal continuation indicator 24
Literal delimiter
Separator 84
Literal-phrase
SPECIAL-NAMES paragraph 220
Literals
Alphanumeric 76
Boolean 78
National 79
Numeric 78
Local names 107, 124
Local time format 615
Locale 66
Category names 66
Field names 66
LOCALE clause 123
SPECIAL-NAMES paragraph 219
Locale field names 67
Locale identification 419, 420
LOCALE phrase 123, 420, 421, 517
PICTURE clause 341
SET statement, locale 558
SPECIAL-NAMES paragraph 220
LOCALE-COMPARE function 667
LOCALE-DATE function 668
Locale-name 69
Definition 72
OBJECT-COMPUTER paragraph 216
Scope of 107
SET statement, locale 558
SPECIAL-NAMES paragraph 219
Locale-names 219
LOCALE-TIME function 669
LOCALE-TIME-FROM-SECONDS function 670
Local-storage section 264
LOCK MODE clause 247
FILE-CONTROL paragraph 236
LOCK ON MULTIPLE clause
LOCK MODE clause 247
LOCK phrase
CLOSE statement 462
READ statement 535
REWRITE statement 547
WRITE statement 602
LOG function 671
LOG10 function 672
Logical file 165
Logical operators 130
In complex conditions 153
Logical record 295, 752
LOWER-CASE function 673
Lower-case letters 63
LOWEST-ALGEBRAIC function 674
LOWLIGHT clause 330
LOWLIGHT phrase
SET statement, attribute setting 558
LOW-VALUE/LOW-VALUES figurative constant 83

M

MANUAL phrase
LOCK MODE clause 247
Mapping letters 747
Matching
Variable-length groups] 119
MAX function 675
Maximum capacity
Exceeding 117
Maximum capacity of a dynamic table
Definition 115
MEAN function 676
MEDIAN function 677

Merge file 176
MERGE statement 508, 545
Merging 755
Meta-terms 20
Method 715
    Invocation 181
Method definition 180
Method overlaoding 180
METHOD phrase
    EXIT statement 11, 481
Method prototypes 185, 196
Method resolution signature
    Definition 180
Method-definition 196
METHOD-ID paragraph 204
Method-name 69, 72
    METHOD-ID paragraph 204
    Scope of 109
Methods 180
MIDRANGE function 678
MIN function 679
Minutes subfield 614
Mnemonic-name 69, 72, 219
    ACCEPT statement 443
    DISPLAY statement 470
    Scope of 107
    SET statement, switch setting 557
    SPECIAL-NAMES paragraph 219
    WRITE statement 602
MOD function 680
mon_decimal_point 67
mon_grouping 67
mon_thousands_sep 67
Month subfield 612
MOVE
    Dynamic-capacity tables 116
MOVE statement 397, 435, 471, 513, 591
    CORRESPONDING phrase 435
    Implicit 494
Multilingual support 798
Multiplication 135
MULTIPLY statement 435, 521

N

N PICTURE symbol 341
n_cs_precedes 67
Names 775
NATIONAL 319
National category 120, 121, 319, 344
National character 14
National character set 61, 793
National class 120
National coded character set 61, 793
National data item 121
National functions 610
National group item 319
National literal 79
    Continuation of 26, 27
NATIONAL phrase
    CODE-SET clause 298
    COLLATING SEQUENCE clause 244
    INITIALIZE statement 493
    MERGE statement 508
    SORT statement 568
    SPECIAL-NAMES paragraph 219, 220
National-edited category 120, 121, 344
National-edited data item 121
NATIONAL-EDITED phrase
    INITIALIZE statement 493
NATIONAL-OF function 681
Native arithmetic 52, 131, 132, 435, 521, 585, 618
Native arithmetic flagging 52
NATIVE phrase
    ARITHMETIC clause 207
    SPECIAL-NAMES paragraph 220
Natural language text 21
Natural logarithm 643, 671
NEAREST-AWAY-FROM-ZERO phrase
    DEFAULT ROUNDED clause 208
    INTERMEDIATE ROUNDING clause 209
    ROUNDED phrase 432
NEAREST-EVEN phrase
    DEFAULT ROUNDED clause 208
    INTERMEDIATE ROUNDING clause 209
    ROUNDED phrase 432
NEAREST-TOWARD-ZERO phrase
    DEFAULT ROUNDED clause 208
    ROUNDED phrase
        PROHIBITED phrase
            ROUNDED phrase 432
Negated conditions 153
negative_sign 67
NEGATIVE-INFINITY phrase
    SET statement 559
NESTED phrase 455
Nested statement 416
New method 715
Next executable statement 419
NEXT GROUP clause 331
NEXT PAGE phrase 327
    NEXT GROUP clause 331
NEXT phrase
    READ statement 535
NEXT RECORD phrase
    READ statement 535
NEXT SENTENCE phrase
    IF statement 491
    SEARCH statement 552
NEXT STATEMENT phrase
    RESUME statement 544
NO ADVANCING phrase
    DISPLAY statement 470
NO LOCK phrase
    READ statement 535

REWRITE statement 547
WRITE statement 602
NO OTHER phrase
OPEN statement 523
SHARING clause 254
NO REWIND phrase
CLOSE statement 462
OPEN statement 523
Non fatal exception conditions 424
Noncontiguous data items 112
Noncontiguous elementary items 272
Nonstandard extension flagging 5
NORMAL phrase
STOP statement 579
Normal run unit termination 422
Normalized values 133
NOT AT END phrase
READ statement 535
RETURN statement 545
NOT AT END-OF-PAGE phrase
WRITE statement 602
NOT AT EOP phrase
WRITE statement 602
NOT INVALID KEY phrase
DELETE statement 468
READ statement 535
REWRITE statement 547
START statement 575
WRITE statement 602
NOT ON EXCEPTION phrase
ACCEPT statement 443
CALL statement
EXCEPTION phrase
CALL statement 454
DISPLAY statement 470
NOT ON OVERFLOW phrase
STRING statement 580
UNSTRING statement 589
NOT ON SIZE ERROR phrase 434
ADD statement 449
COMPUTE statement 465
DIVIDE statement 473
MULTIPLY statement 521
SUBTRACT statement 583
NOT operator
In complex conditions 153
In negated conditions 153
NOT phrase
EVALUATE statement 476
NOT-A-NUMBER phrase
SET statement 559
NULL
Predefined object 99
Predefined-address 101
NULL phrase
SET statement, pointer assignment 558
NULL predefined address 485
ALLOCATE statement 452
INITIALIZE statement 495
NULL predefined object reference
INITIALIZE statement 495
Numeric category 120, 122, 344
Numeric character 14
Numeric class 120
Numeric comparison 145
Numeric data item 122
Numeric functions 610
Numeric literal 78
Fixed-point 78
Floating-point 78
NUMERIC phrase
CLASS clause 149, 296
INITIALIZE statement 493
NUMERIC-EDITED
INITIALIZE statement 493
Numeric-edited category 122, 293, 344
Category of data
Numeric-edited 120
Numeric-edited data item 122
NUMVAL function 682
NUMVAL-C function 683
NUMVAL-F function 685

O

OBJECT 593
Object
Instance 189
Interface 185
Life cycle of 189
OBJECT paragraph 206
Object class 120
Object data item 817
Object orientation
Class 180
Class-name 71
class-name 68
Object oriented concepts 816
OBJECT paragraph 206
Object properties 100, 204
Object references 180
Initialization 404, 453
OBJECT-COMPUTER paragraph 216
Object-reference category 120, 122, 395
Object-reference data item 122
OBJECT-REFERENCE phrase
INITIALIZE statement 493
Objects and classes 180
Object-view 98
Obsolete language element flagging 5
Obsolete language elements 5, 874
ARITHMETIC IS STANDARD 874
FLAG-85 directive 874
MOVE of QUOTES to numeric items 874
OCCURS clause 89, 333, 366, 370, 381, 397, 402, 407,

552, 569, 598
Occurs-depending group item
Definition 337
Occurs-depending tables
Definition 115
OF phrase
COPY statement 33
With identifiers 90
With qualification 88
OFF phrase
SET statement, attribute setting 558
SET statement, switch setting 557
SPECIAL-NAMES paragraph 219
Offset subformat 615
Offset time format 615
Offset-hours subfield 615
Offset-minutes subfield 615
OMITTED phrase 93, 504
CALL statement 455
Omitted-argument condition 152
ON EXCEPTION phrase
ACCEPT statement 443
CALL statement 454
DISPLAY statement 470
ON OVERFLOW phrase
CALL statement 454
STRING statement 580
UNSTRING statement 589
ON phrase
SET statement, attribute setting 558
SET statement, switch setting 557
SPECIAL-NAMES paragraph 219
VALIDATE-STATUS clause 398
ON SIZE ERROR phrase 433
ADD statement 449
COMPUTE statement 465
DIVIDE statement 473
MULTIPLY statement 521
SUBTRACT statement 583
ONLY phrase 439
USAGE clause 391
Open mode 166, 523, 753
OPEN statement 523
Opening delimiter 84
Operands 18
Operational sign 112
Operators
Arithmetic 128
Boolean 128
Concatenation 128
Invocation 128
Logical 130
Relational 128
OPTIONAL phrase
FILE-CONTROL paragraph 236
Procedure division header 93, 410
Optional words 18, 75
OPTIONS paragraph 207
OR operator
In combined conditions 153
In complex conditions 153
OR PAGE phrase
TYPE clause 384
OR phrase
UNSTRING statement 589
ORD function 686
ORDER keyword
SORT statement 568
Order of evaluation of conditions 155
ORDER TABLE 219, 700
Ordering-name 69, 72, 219
Scope of 107
Ordinal date format 613
ORD-MAX function 687
ORD-MIN function 688
Organization 166
Indexed 167
Relative 167
Sequential 167
ORGANIZATION clause 249
FILE-CONTROL paragraph 236
OTHER phrase
EVALUATE statement 476
SELECT WHEN clause 373
Out-of-line PERFORM statement 528
OUTPUT phrase
OPEN statement 523
USE statement 593
OUTPUT PROCEDURE phrase
MERGE statement 508
SORT statement 568
OVERFLOW phrase
CALL statement 454
STRING statement 580
UNSTRING statement 589
Overlapping operands 421, 496
STRING statement 582
OVERRIDE clause
METHOD-ID paragraph 204

P

P PICTURE symbol 341
p_cs_precedes 67
PACKED-DECIMAL phrase
USAGE clause 391
Page advance 487
PAGE clause 339
PAGE directive 56
Page fit processing 486
PAGE FOOTING phrase
TYPE clause 384
PAGE HEADING phrase
TYPE clause 384
PAGE LIMIT clause

Report description entry 273
PAGE phrase
WRITE statement 602
PAGE-COUNTER 105, 109, 273
Qualified 88
Paragraph header 199
PARAGRAPH phrase
EXIT statement 11, 481
Paragraph-name 69, 72, 413
Qualified 88
Paragraphs 199, 413
Parameterized classes 188
Parameterized interfaces 189
Parameter-name 69, 72
CLASS-ID paragraph 200
Parametric polymorphism 180
Parentheses
Function-identifier 93
In arithmetic expressions 131
Separator 84
Parenthesis
In Boolean expression 140
In logical conditions 153
PERFORM phrase
EXIT statement 11, 481
PERFORM statement 528
PERFORM statement examples 853
Period
Separator 20, 84
PF phrase
TYPE clause 384
PH phrase
TYPE clause 384
PHYSICAL 632, 665
Physical file 165
Physical record 295
Physical-structure-name 74
PI function 689
PICTURE character-string 580
Picture character-string 580
Picture character-strings 84
PICTURE clause 341, 392, 397
Data description entry 276
Precedence rules 352
PICTURE SYMBOL phrase
SPECIAL-NAMES paragraph 219
Pointer 456
Initialization 404, 453
Pointer class 120
POINTER phrase
STRING statement 580
UNSTRING statement 589
USAGE clause 391
Polymorphism 180
Class 180
Parametric 180
positive_sign 67
POSITIVE-INFINITY phrase
SET statement 559
Precedence of logical operators 153
Predefined object reference 597
Predefined-address 101
PREFIXED phrase
ANY LENGTH STRUCTURE clause 220
PRESENT WHEN clause 301, 318, 326, 356, 365, 598
PRESENT-VALUE function 690
Prime record key 251
PRINTING phrase
SUPPRESS statement 586
Procedure division 409
Declarative portion 412
Nondeclarative portion 412
PROCEDURE DIVISION header 409
Procedure-name 413
Procedures 413
Processor-dependent element flagging 4
Processor-dependent language elements 4
List of 736
PROGRAM COLLATING SEQUENCE clause
OBJECT-COMPUTER paragraph 216
PROGRAM phrase
EXIT statement 11, 481
Program prototype
CALL statement 455
CANCEL statement 460
Program-address-identifier 104
Program-definition 193
PROGRAM-ID paragraph 211
Program-name 69, 72
PROGRAM-ID paragraph 211
Scope of 109
Program-pointer
CANCEL statement 460
Definition 395
Program-pointer category 104, 120, 122
Program-pointer data item 122
PROGRAM-POINTER phrase
INITIALIZE statement 493
USAGE clause 391
Program-prototype 193
Program-prototype-name 69, 72
PROHIBITED phrase
DEFAULT ROUNDED clause 208
INTERMEDIATE ROUNDING clause 209
PROPAGATE directive 57
PROPERTY clause 100, 358
METHOD-ID paragraph 204
Property-name 69, 72
PROTOTYPE clause
FUNCTION-ID paragraph 202
PROGRAM-ID paragraph 211
Prototype versus repository flagging 164
Pseudo-text 32, 37
Pseudo-text delimiter 32
Separator 85

Q

Qualification 90
  Implicit 86
  LINAGE-COUNTER 88
  LINE-COUNTER 88
  Of condition-names 106
  Of data-names 91
  Of paragraph-names 88
  OF phrase 88
  PAGE-COUNTER 88
  Subscripting 90
  Uniqueness of reference 86
Qualified 91
Quotation mark 63
Quotation symbol 84
QUOTE/QUOTES figurative constant 83

R

RAISE statement 534
RAISING phrase 489
  EXIT statement 481
  Procedure division header 409
Random access mode 168, 753
RANDOM function 691
RANDOM phrase
  ACCESS MODE clause 236, 242
RANGE function 692
Range of PERFORM statement 532
RD entry 267
READ ONLY phrase
  OPEN statement 523
  SHARING clause 254
READ statement 535, 548
Receiving operand 115, 420, 436, 455, 482, 506, 534, 558
Record
  Logical 295
  Physical 295
RECORD clause 361, 576
RECORD CONTAINS clause
  File description entry 259
RECORD DELIMITER clause 250
  FILE-CONTROL paragraph 238
Record description entry 272
RECORD IS VARYING clause 606
Record key 243, 251
RECORD KEY clause 251
RECORD keyword
  RETURN statement 545
  REWRITE statement 547
  UNLOCK statement 588
Record locking 174, 175, 247, 254, 588, 757
Record locks 576, 588
Record operations 752
RECORD phrase
  DELETE statement 468
Record selection 373
RECORD-KEY clause
  FILE-CONTROL paragraph 236
Record-key-name 69, 73
  RECORD KEY clause 251
Record-name 69, 73
  Scope of 108
RECORDS keyword
  UNLOCK statement 588
RECORDS phrase
  BLOCK CONTAINS clause 295
RECURSIVE clause
  PROGRAM-ID paragraph 211
Recursive functions 189
Recursive methods 180
Recursive program 126
REDEFINES clause 364, 370, 381, 401, 494
Reel 168
REEL phrase
  CLOSE statement 462
Reference format 23
  Fixed-form 25
  Free-form 27
  Logical conversion 28
Reference format example 857
REFERENCE phrase
  INVOKE statement 504
Reference-modification 95
References 86
Relation condition
  Alphanumeric operands 145
  Arithmetic expression in 144
  Boolean operands 146
  Data-pointer operands 148
  Index data item in 144
  Index data items 147
  Mixed operands 145
  National operands 146
  Numeric operands 145
  Object reference identifiers 148
  Order of evaluation 155
  Strongly-typed group items 147
RELATION phrase
  VALIDATE-STATUS clause 398
Relation validation 598
Relational operator 128
  START statement 575
Relative files 167
RELATIVE KEY clause 252
  FILE-CONTROL paragraph 237
  START statement 575
Relative organization 751
RELATIVE phrase
  ORGANIZATION clause 249
Relative record number 167, 252
RELEASE statement 542
REM function 693
REMAINDER phrase

DIVIDE statement 473
REMOVAL phrase
CLOSE statement 462
RENAMES clause 112, 366, 494
REPEATED phrase
VALUE clause 400
REPLACE statement 31, 37, 47, 53
REPLACING phrase
COPY statement 33
INITIALIZE statement 493
INSPECT statement 498
REPORT clause 367
Report description entry 267, 272
REPORT FOOTING phrase
TYPE clause 384
Report group description entry 273
REPORT HEADING phrase
TYPE clause 384
Report section 267
Report writer 835
Report-name 69, 73, 367
Report description entry 273
Scope of 108
Repository 164
REPOSITORY paragraph 68, 230, 594
REPRESENTS-NOT-A-NUMBER phrase
CLASS clause 149
REQUIRED clause 368
Required word 75
RESERVE clause 253
FILE-CONTROL paragraph 236
Reserved word 68
Implementation 4
Reserved words 74
Compiler directives 163
Source text 156
RESET phrase
NEXT GROUP clause 331
SUM clause 377
Restricted data pointer 103
Restricted data-pointer
Definition 395
Restricted pointer 396
RESUME statement 544
RETRY phrase 437, 527, 760
DELETE statement 468
OPEN statement 523
READ statement 535
REWRITE statement 547
WRITE statement 602
RETURN statement 545
Returned values, intrinsic functions 618
RETURNING phrase
ALLOCATE statement 452
CALL statement 454
INVOKE statement 504
Procedure division header 94, 409
REVERSE function 694
REVERSE-VIDEO clause 369
REVERSE-VIDEO phrase
SET statement, attribute setting 558
REWRITE statement 547
RF phrase
TYPE clause 384
RH phrase
TYPE clause 384
RIGHT phrase
JUSTIFIED clause 322
SYNCHRONIZED clause 381
ROUNDED phrase 133, 134, 432, 436
ADD statement 449
COMPUTE statement 465
DIVIDE statement 473
MULTIPLY statement 521
SOURCE clause 375
SUBTRACT statement 583
SUM clause 377
Rounding
Concepts 805
Rounding rules 133
Rules 20
Run unit 16, 416
Run unit termination 422, 579
Runtime element 192

S

S PICTURE symbol 341, 374
SAME AREA clause 255, 509, 569
SAME AS clause 370
SAME clause 255
SAME RECORD AREA clause 255, 542, 548, 604
SAME SORT AREA clause 255, 509, 569
SAME SORT-MERGE AREA clause 255, 509, 569
Scope of names 106
Scope of statements 415
Scope terminators 415
Screen description entry 281
Screen section 269
Screen-name 69, 73, 309
Screen-names
Scope of 108
Screens 177
SD entry 262
SEARCH statement 392, 552
SECONDS phrase
DELETE statement 468
OPEN statement 523
READ statement 535
RETRY phrase 437
REWRITE statement 547
WRITE statement 602
Seconds subfield 614
SECONDS-FROM-FORMATTED-TIME function 695
SECONDS-PAST-MIDNIGHT function 696

SECTION header 409
SECTION phrase
EXIT statement 11, 481
Section-name 69, 73, 413
Sections 413
SECURE clause 372
SELECT clause
FILE-CONTROL paragraph 236
SELECT WHEN clause 370, 373
SORT statement 571
SELF predefined object identifier 99
Semicolon 20
Interchangeable with comma 20, 84
Separator 84
Sending operand 115, 420, 436, 455, 482, 506, 534, 558
Sentence 413
Definition 413
SEPARATE CHARACTER phrase
SIGN clause 374
Separators 20, 84
Colon 85
Comma 84
Literal delimiter 84
Parentheses 84
Period 20, 84
Pseudo-text delimiter 85
Semicolon 84
Space 84
Sequence number area 25
Sequential access mode 168, 753
Sequential files 167
Sequential organization 751
SEQUENTIAL phrase
ACCESS MODE clause 236, 242
ORGANIZATION clause 249
SET statement 392, 557
Implicit 494
Set statement
Dynamic-capacity tables 115
Shared files 174, 527
Shared memory area 768
SHARING clause 247, 254, 510, 572
FILE-CONTROL paragraph 236
Sharing data 126
Sharing file connectors 166
Sharing mode 174, 756
SHARING phrase 247
MERGE statement 510
OPEN statement 523
SORT statement 572
SHORT phrase
ANY LENGTH STRUCTURE clause 220
SIGN clause 112, 371, 374, 384
Sign condition 152
SIGN function 697
Signature 164
Method resolution 180
SIGNED phrase
ANY LENGTH STRUCTURE clause 220
USAGE clause 391
Simple condition 142
Simple insertion editing 349
Simple relational operator 129
SIN function 698
Size error condition 136, 433
SIZE ERROR phrase 433
ADD statement 449
COMPUTE statement 465
DIVIDE statement 473
MULTIPLY statement 521
SUBTRACT statement 583
SIZE phrase
PICTURE clause 341
STRING statement 580
Slant (/) comment line 27
Sort file 176
SORT statement 542, 545, 568
Sorting 754
Sorting tables 765
Sort-merge file description entry 262
SOURCE clause 375
Source element 192
SOURCE FORMAT directive 58
Source indicator 24
SOURCE phrase
ALTERNATE RECORD KEY clause 243
RECORD KEY clause 251
Source unit 191
SOURCE-COMPUTER paragraph 215
Space
Separator 84
SPACE/SPACES figurative constant 83
Special character words 75
In formats 20
Special insertion editing 349
SPECIAL-NAMES paragraph 150, 151, 219, 562
ACCEPT statement 444
SQRT function 699
Standard arithmetic 52, 131, 132, 436, 522, 618
Concepts 808
NEAREST-AWAY-FROM-ZERO phrase of INTERMEDIATE ROUNDING clause 133
NEAREST-EVEN phrase of INTERMEDIATE ROUNDING clause 134
PROHIBITED phrase of INTERMEDIATE ROUNDING clause 134
TRUNCATION phrase of INTERMEDIATE ROUNDING clause 134
Standard classes 715
Standard date form 619
Standard intermediate data item 132, 133
Standard language element acceptance 3
Standard numeric time form 619
STANDARD phrase
ARITHMETIC clause 207
STANDARD-1

FILE-CONTROL paragraph 238
STANDARD-1 phrase
RECORD DELIMITER clause 250
STANDARD-1,2 phrase
SPECIAL-NAMES paragraph 220
Standard-binary
Arithmetic 136
Standard-binary arithmetic 136, 436, 618
Standard-binary intermediate data item 136
STANDARD-BINARY phrase
ARITHMETIC clause 207
STANDARD-COMPARE function 700
Standard-decimal
Arithmetic 138
Standard-decimal arithmetic 138, 436, 618
Standard-decimal intermediate data item 138
STANDARD-DECIMAL phrase
ARITHMETIC clause 207
STANDARD-DEVIATION function 701
START statement 575
State of
Function 416
Method 416
Object data 418
Program 416
Statement 413
Compiler-directing 31
Declarative 413
Procedural 413
Static data 125, 263
Static items 125
STOP 419
STOP RUN statement 579
STOP statement 489, 579
STRING statement 580
STRONG phrase
TYPEDEF clause 389
Strongly-typed group items 122
Strongly-typed groups 122
Strongly-typed items 785
Structured compilation group 30, 31, 191
Structured constant 304, 842
Subfield 612
Subscripted
Data-name 89
Subscripted identifier 90
Subscripting 762
Condition-name 89
Subscripts 88, 89
Dynamic-capacity tables 115
Substantive changes list 865
SUBTRACT statement 435, 583
CORRESPONDING phrase 435
Subtraction 134
SUM clause 377
Sum counter 378
SUM function 702
Summary of functions 619
SUPER predefined object identifier 99
SUPPRESS statement 586
Surrogate pair 97
Definition 16
Switch status
Condition-name 106
Switch-name 74, 151
SPECIAL-NAMES paragraph 219
Switch-status condition 151
Symbolic-character 69, 73, 219
Figurative constant 84
Scope of 107
SYMBOLIC-CHARACTERS clause
SPECIAL-NAMES paragraph 220
SYNCHRONIZED clause 114, 381, 401
Syntax rules 20
Syntax violation flagging 3
SYSTEM-DEFAULT phrase
SET statement 558
System-name 73
System-names 219

T

t_fmt 67
Table handling 760
Table sort 568
Tables 114, 333
TALLYING phrase
INSPECT statement 498
UNSTRING statement 589
TAN function 703
Terminal screen 177
TERMINATE statement 587, 594
TEST AFTER phrase
PERFORM statement 529
TEST BEFORE phrase
PERFORM statement 529
TEST phrase
PERFORM statement 528
TEST-FORMATTED-DATETIME function 706
TEST-NUMVAL function 707
Text manipulation 30
Text manipulation stage 30
Text-name 34, 74
Text-word 32
THEN phrase
IF statement 491
INITIALIZE statement 493
THEN REPLACING phrase
INITIALIZE statement 493
THROUGH (THRU) phrase 436
EVALUATE directive 46
EVALUATE statement 476
MERGE statement 508
PERFORM statement 528
RENAMES clause 366

SORT statement 568
VALUE clause 400
Time format 613
TIME phrase
ACCEPT statement 443
TIMES phrase
DELETE statement 468
OPEN statement 523
PERFORM statement 528
READ statement 535
RETRY phrase 437
REWRITE statement 547
VALUE clause 400
WRITE statement 602
TO clause 383
TO END phrase
VALUE clause 400
TO phrase
ADD statement 449
INITIALIZE statement 493
INSPECT statement 498
MOVE statement 513
SET statement, condition setting 557
SET statement, index assignment 557
SET statement, object identifier assignment 557
SET statement, pointer assignment 558
SET statement, switch setting 557
TO VALUE clause
INITIALIZE statement 493
TOWARD-GREATER phrase
DEFAULT ROUNDED clause 208
ROUNDED phrase 432
TOWARD-LESSER phrase
DEFAULT ROUNDED clause 208
ROUNDED phrase 432
TRAILING phrase
SIGN clause 374
Transfer of control 418
EXIT FUNCTION statement 419
EXIT METHOD statement 419
EXIT PROGRAM statement 419
EXIT statement 419
GOBACK statement 419, 489
MERGE statement 418
PERFORM statement 419
SORT statement 418
STOP statement 419
TRIM function 710
TRUE phrase
EVALUATE directive 46
EVALUATE statement 476
SET statement, condition setting 557
TRUNCATION phrase
DEFAULT ROUNDED clause 208
INTERMEDIATE ROUNDING clause 209
ROUNDED phrase 432
TURN directive 59
TYPE clause 370, 384
TYPEDEF clause 389
Data description entry 276
Type-name 69, 73
Scope of 108
TYPE clause 384
Types
Definition of 122
Types of functions 610

U

UCS 9, 10, 16, 61, 77, 97, 515, 793, 803
Definition 16
UCS-4 226
UCS-4 phrase
SPECIAL-NAMES paragraph 220
Unary minus 136
Unary plus 136
Undefined language element list 731
UNDERLINE clause 390
UNDERLINE phrase
SET statement, attribute setting 558
Underscore 68
Unicode 803
Uniqueness of reference 86
Unit 168
UNIT phrase
CLOSE statement 462
Universal object reference 395
UNLOCK statement 588
UNSIGNED phrase
USAGE clause 391
UNSTRING statement 589
UNTIL phrase
PERFORM statement 528
UP phrase
SET statement, index arithmetic 557, 559
SET statement, pointer arithmetic 558
UPON phrase
DISPLAY statement 470
SUM clause 377
UPPER-CASE function 711
Upper-case letters 63
USAGE clause 371, 384, 391
USE procedures 461
USE statement 412, 527, 593
User default locale 66
USER-DEFAULT phrase
SET statement 558
User-defined functions 189, 230
User-defined words 743
User-function-name 69, 73
USING clause 397
CLASS-ID paragraph 200
INTERFACE-ID paragraph 203
USING phrase
CALL statement 454

FILE-CONTROL paragraph 236
INVOKE statement 504
MERGE statement 508
Procedure division header 410
SORT statement 568
UTC time format 615
UTF-16 9, 16, 112
SPECIAL-NAMES paragraph 220
UTF-8 77
SPECIAL-NAMES paragraph 220

V

V PICTURE symbol 341
VALID phrase
VALUE clause 400
Validate facility 843
VALIDATE statement 321, 407, 598
VALIDATE-STATUS clause 398, 598
VALUE clause 364, 400, 453, 495
VALUE phrase
INITIALIZE statement 493
INVOKE statement 504
Variable-length data item
Contiguity of data items 118
Definition 118
Variable-length group
Comparison of 148
Variable-length groups
Compatibility 118
Definition 118
Variable-length records 752
VARIANCE function 712
VARYING clause 407
VARYING phrase
PERFORM statement 529
RECORD clause 361
SEARCH statement 552

W

Weakly-typed items 123, 785
Week date format 613
Week-of-year subfield 613
WHEN phrase
EVALUATE directive 46
EVALUATE statement 476
INVALID clause 321
SEARCH statement 552
WHEN SET TO FALSE phrase
VALUE clause 400
WHEN-COMPILED function 713
WITH DUPLICATES phrase
SORT statement 568
WITH ERROR STATUS phrase
STOP statement 579
WITH FILLER phrase
INITIALIZE statement 493
WITH FOOTING phrase
LINAGE clause 324
WITH LENGTH phrase
START statement 575
WITH LOCK phrase
CLOSE statement 462
READ statement 535
REWRITE statement 547
WRITE statement 602
WITH NO ADVANCING phrase
DISPLAY statement 470
WITH NO LOCK phrase
READ statement 535
REWRITE statement 547
WRITE statement 602
WITH NO REWIND phrase
CLOSE statement 462
OPEN statement 523
WITH NORMAL STATUS phrase
STOP statement 579
WITH POINTER phrase
STRING statement 580
UNSTRING statement 589
WITH TEST phrase
PERFORM statement 528
Word 68
Working-storage section 263
WRITE statement 221, 602

X

X PICTURE symbol 341

Y

Year subfield 612
YEAR-TO-YYYY function 714
YYYYDDD phrase
ACCEPT statement 443
YYYYMMDD phrase
ACCEPT statement 443

Z

Z PICTURE symbol 341
ZERO/ZEROES/ZEROS figurative constant 83
Zero-length item 123, 337, 363
Zero-length literal
Definition 76

