# Language Reference

SC26-4769-03

# IBM COBOL Set for AIX
# IBM COBOL for MVS & VM
# IBM VisualAge for COBOL for OS/2

IBM

IBM

# Language Reference

SC26-4769-03

## IBM COBOL Set for AIX
## IBM COBOL for MVS & VM
## IBM VisualAge for COBOL for OS/2

**Note!**

Before using this information and the product it supports, be sure to read the general information under “Notices” on page x.

**Fourth Edition (September 1996)**

This edition replaces and makes obsolete the previous edition, SC26-4769-02. The technical changes for this edition are summarized under "Summary of Changes," and are indicated by a vertical bar to the left of a change. This edition is available in softcopy format only.

This edition applies to:

IBM COBOL for MVS & VM Release 2 (Program Number 5688-197)
IBM COBOL Set for AIX Release 1 (Program Number 5765-548)
IBM VisualAge for COBOL for OS/2 Release 2 (Program Number 5622-793), including IBM COBOL support for Windows NT and Windows 95

and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, W92/H3
P.O. Box 49023
San Jose, California, 95161-9023
U.S.A.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1991, 1996. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

# Contents


**Notices** . . . . . x
Programming Interface Information . . . . . x
Trademarks . . . . . xi

**About This Book** . . . . . xii
IBM Extensions . . . . . xii
Obsolete Language Elements . . . . . xii
How to Read the Syntax Diagrams . . . . . xiii
DBCS Notation . . . . . xv
Acknowledgment . . . . . xvi

**Summary of Changes** . . . . . xvii
September 1996 . . . . . xvii
July 1996 . . . . . xvii
October 1995 . . . . . xviii
Extensions for Object-Oriented COBOL (MVS, AIX, OS/2, and Windows Only) xviii
Extensions for Interoperability (MVS, VM, AIX, and OS/2) . . . . . xviii
Support for COBOL on AIX and OS/2 . . . . . xix

**Part 1. COBOL Language Structure** . . . . . 3

**Characters** . . . . . 4
Character-Strings . . . . . 5
Figurative Constants . . . . . 10
Special Registers . . . . . 12
Literals . . . . . 22
Separators . . . . . 30

**Sections and Paragraphs** . . . . . 32
Statements and Clauses . . . . . 32

**Reference Format** . . . . . 34
Sequence Number Area . . . . . 34
Indicator Area . . . . . 34
Area A . . . . . 35
Area B . . . . . 37
Area A or Area B . . . . . 39

**Scope of Names** . . . . . 41
Types of Names . . . . . 41
External and Internal Resources . . . . . 44
Resolution of Names . . . . . 45

**Referencing Data Names, Copy Libraries, and Procedure Division Names** . 46


© Copyright IBM Corp. 1991, 1996

Uniqueness of Reference . . . . . 46

**Transfer of Control** . . . . . 58

**Part 2. COBOL Source Unit Structure** . . . . . 61

**COBOL Program Structure** . . . . . 62
Nested Programs . . . . . 65

x **COBOL Class Definition Structure** . . . . . 68

x **COBOL Method Definition Structure** . . . . . 70

**Part 3. Identification Division** . . . . . 71

**Identification Division** . . . . . 72
PROGRAM-ID Paragraph . . . . . 75
x CLASS-ID Paragraph . . . . . 77
x METHOD-ID Paragraph . . . . . 79
Optional Paragraphs . . . . . 81

**Part 4. Environment Division** . . . . . 83

**Configuration Section** . . . . . 84
SOURCE-COMPUTER Paragraph . . . . . 85
OBJECT-COMPUTER Paragraph . . . . . 86
SPECIAL-NAMES Paragraph . . . . . 88
ALPHABET Clause . . . . . 91
SYMBOLIC CHARACTERS Clause . . . . . 93
CLASS Clause . . . . . 94
CURRENCY SIGN Clause . . . . . 95
x REPOSITORY Paragraph . . . . . 96

**Input-Output Section** . . . . . 98
FILE-CONTROL Paragraph . . . . . 99
SELECT Clause . . . . . 103
ASSIGN Clause . . . . . 103
RESERVE Clause . . . . . 107
ORGANIZATION Clause . . . . . 108
PADDING CHARACTER Clause . . . . . 111
RECORD DELIMITER Clause . . . . . 111
ACCESS MODE Clause . . . . . 112
RECORD KEY Clause . . . . . 114
ALTERNATE RECORD KEY Clause . . . . . 115
RELATIVE KEY Clause . . . . . 116
x PASSWORD Clause . . . . . 116

x LOCK MODE Clause . . . 117
FILE STATUS Clause . . . 119
I-O-CONTROL Paragraph . . . 120
RERUN Clause . . . 122
SAME AREA Clause . . . 123
SAME RECORD AREA Clause . . . 124
SAME SORT AREA Clause . . . 125
SAME SORT-MERGE AREA Clause . . . 125
MULTIPLE FILE TAPE Clause . . . 126
APPLY WRITE-ONLY Clause . . . 126

**Part 5. Data Division** . . . 127

**Data Division Overview** . . . 128
File Section . . . 129
Working-Storage Section . . . 129
x Local-Storage Section . . . 131
Linkage Section . . . 131
Data Types . . . 132
Data Relationships . . . 133

**Data Division—File Description Entries** . . . 140
File Section . . . 144
EXTERNAL Clause . . . 144
GLOBAL Clause . . . 145
BLOCK CONTAINS Clause . . . 145
RECORD Clause . . . 147
LABEL RECORDS Clause . . . 150
VALUE OF Clause . . . 151
DATA RECORDS Clause . . . 151
LINAGE Clause . . . 151
x RECORDING MODE Clause . . . 154
CODE-SET Clause . . . 156

**Data Division—Data Description Entry** . . . 157
Format 1 . . . 157
Format 2 . . . 158
Format 3 . . . 158
Level-Numbers . . . 158
BLANK WHEN ZERO Clause . . . 160
EXTERNAL Clause . . . 160
GLOBAL Clause . . . 161
JUSTIFIED Clause . . . 162
OCCURS Clause . . . 163
PICTURE Clause . . . 169
REDEFINES Clause . . . 186
RENAMES Clause . . . 190
SIGN Clause . . . 192

SYNCHRONIZED Clause . . . . . 193
USAGE Clause . . . . . 200
VALUE Clause . . . . . 207

**Part 6. Procedure Division** . . . . . 213

**Procedure Division Structure** . . . . . 215
x Requirements for a Method Procedure Division . . . . . 216
The Procedure Division Header . . . . . 217
Declaratives . . . . . 220
Procedures . . . . . 221
Arithmetic Expressions . . . . . 223
Conditional Expressions . . . . . 225
Statement Categories . . . . . 245
Statement Operations . . . . . 249

**Procedure Division Statements** . . . . . 261
ACCEPT Statement . . . . . 261
ADD Statement . . . . . 266
ALTER Statement . . . . . 268
CALL Statement . . . . . 270
CANCEL Statement . . . . . 277
CLOSE Statement . . . . . 279
COMPUTE Statement . . . . . 283
CONTINUE Statement . . . . . 285
DELETE Statement . . . . . 286
DISPLAY Statement . . . . . 288
DIVIDE Statement . . . . . 291
ENTRY Statement . . . . . 294
EVALUATE Statement . . . . . 295
EXIT Statement . . . . . 299
x EXIT METHOD Statement . . . . . 300
EXIT PROGRAM Statement . . . . . 301
GOBACK Statement . . . . . 302
GO TO Statement . . . . . 303
IF Statement . . . . . 305
INITIALIZE Statement . . . . . 307
INSPECT Statement . . . . . 310
x INVOKE Statement . . . . . 319
MERGE Statement . . . . . 327
MOVE Statement . . . . . 333
MULTIPLY Statement . . . . . 338
OPEN Statement . . . . . 340
PERFORM Statement . . . . . 346
READ Statement . . . . . 358
RELEASE Statement . . . . . 366
RETURN Statement . . . . . 368
REWRITE Statement . . . . . 370

SEARCH Statement . . . . . 374
SET Statement . . . . . 381
SORT Statement . . . . . 388
START Statement . . . . . 395
STOP Statement . . . . . 398
STRING Statement . . . . . 400
SUBTRACT Statement . . . . . 405
UNSTRING Statement . . . . . 407
WRITE Statement . . . . . 415

**Part 7. Intrinsic Functions** . . . . . 425

**Intrinsic Functions** . . . . . 426
Specifying a Function . . . . . 426
Function Definitions . . . . . 432
ACOS . . . . . 435
ANNUITY . . . . . 436
ASIN . . . . . 437
ATAN . . . . . 438
CHAR . . . . . 439
COS . . . . . 440
CURRENT-DATE . . . . . 441
DATE-OF-INTEGER . . . . . 443
DAY-OF-INTEGER . . . . . 444
FACTORIAL . . . . . 445
INTEGER . . . . . 446
INTEGER-OF-DATE . . . . . 447
INTEGER-OF-DAY . . . . . 448
INTEGER-PART . . . . . 449
LENGTH . . . . . 450
LOG . . . . . 451
LOG10 . . . . . 452
LOWER-CASE . . . . . 453
MAX . . . . . 454
MEAN . . . . . 455
MEDIAN . . . . . 456
MIDRANGE . . . . . 457
MIN . . . . . 458
MOD . . . . . 459
NUMVAL . . . . . 460
NUMVAL-C . . . . . 461
ORD . . . . . 463
ORD-MAX . . . . . 464
ORD-MIN . . . . . 465
PRESENT-VALUE . . . . . 466
RANDOM . . . . . 467
RANGE . . . . . 468
REM . . . . . 469

REVERSE . . . . . 470
SIN . . . . . 471
SQRT . . . . . 472
STANDARD-DEVIATION . . . . . 473
SUM . . . . . 474
TAN . . . . . 475
UPPER-CASE . . . . . 476
VARIANCE . . . . . 477
WHEN-COMPILED . . . . . 478

**Part 8. Compiler-Directing Statements** . . . . . 481

**Compiler-Directing Statement** . . . . . 482
x BASIS Statement . . . . . 482
x CBL (PROCESS) Statement . . . . . 483
x *CONTROL (*CBL) Statement . . . . . 484
COPY Statement . . . . . 486
x DELETE Statement . . . . . 493
x EJECT Statement . . . . . 495
ENTER Statement . . . . . 496
x INSERT Statement . . . . . 497
x READY or RESET TRACE Statement . . . . . 498
REPLACE Statement . . . . . 500
x SERVICE LABEL Statement . . . . . 504
x SERVICE RELOAD Statement . . . . . 505
x SKIP1/2/3 Statements . . . . . 506
x TITLE Statement . . . . . 507
USE Statement . . . . . 508

**Compiler Directives** . . . . . 514
x CALLINTERFACE . . . . . 514

**Appendixes** . . . . . 517

**Appendix A. Compiler Limits** . . . . . 518

**Appendix B. EBCDIC and ASCII Collating Sequences** . . . . . 522
EBCDIC Collating Sequence . . . . . 522
US English ASCII Code Page (ISO 646) . . . . . 526

**Appendix C. Source Language Debugging** . . . . . 529
Coding Debugging Lines . . . . . 529
Coding Debugging Sections . . . . . 529
DEBUG-ITEM Special Register . . . . . 530
Activate Compile-Time Switch . . . . . 530
Activate Object-Time Switch . . . . . 530

**Appendix D. Reserved Words** . . . . . 532

**Appendix E. ASCII Considerations for MVS and VM** . . . . . 539
Environment Division . . . . . 539
Data Division . . . . . 541
Procedure Division . . . . . 541

**Appendix F. Locale Considerations (Workstation Only)** . . . . . 543

**Appendix G. Summary of Language Difference: Host COBOL and Workstation COBOL** . . . . . 544

**Appendix H. Industry Specifications** . . . . . 546
Standard Terminology . . . . . 548

**Bibliography** . . . . . 550

**Glossary** . . . . . 552

**Index** . . . . . 574

# Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
555 Bailey Avenue, W92/H3
P.O. Box 49023
San Jose, CA 95161-9023

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

## Programming Interface Information

This book is intended to help you write programs using IBM COBOL for MVS & VM, IBM COBOL Set for AIX, and IBM VisualAge for COBOL for OS/2 compilers. This *IBM COBOL Language Reference* documents General-Use Programming Interface and Associated Guidance Information provided by IBM COBOL for MVS & VM, IBM VisualAge for COBOL for OS/2, and IBM COBOL Set for AIX.

General-Use programming interfaces allow the customer to write programs that obtain the services of IBM COBOL for MVS & VM, IBM VisualAge for COBOL for OS/2, and IBM COBOL Set for AIX.


© Copyright IBM Corp. 1991, 1996

## Trademarks

The following terms are trademarks of the IBM Corporation in the United States and/or other countries or both:

| | |
|---|---|
| AD/Cycle | IBM |
| Advanced Function Printing | IMS |
| AFP | Language Environment |
| AIX | MVS/ESA |
| BookManager | MVS/XA |
| C/370 | Operating System/2 |
| CICS | OS/2 |
| COBOL/370 | Print Services Facility |
| DATABASE 2 | PSF |
| DB2 | SAA |
| DFSMS/MVS | SOMobjects |
| DFSORT | System Object Model |
| ESA/370 | VisualAge |

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

# About This Book

This reference presents the syntax of IBM COBOL for MVS & VM, IBM COBOL Set for AIX, and IBM VisualAge for COBOL for OS/2, including IBM COBOL support for Windows NT and Windows 95. Throughout this book, we use the term IBM COBOL to refer to the three products mentioned above. To indicate platform-specific information, we either:

- Prefix the text with platform-specific indicators (for example, `Under AIX, OS/2, and Windows...`)
- Add parenthetical qualifications (for example, (`Workstation only`)
- Prefix the text with icons. We use the following icons:

  MVS/VM Informs you of information specific to IBM COBOL for MVS & VM.

  WRKSTN Informs you of information specific to IBM COBOL Set for AIX and IBM VisualAge for COBOL for OS/2, including IBM COBOL support for Windows NT and Windows 95.

  AIX Informs you of information specific to IBM COBOL Set for AIX.

**Note:** This book documents extensions for object-oriented COBOL. Object-oriented COBOL **is not** supported on VM.

Use this book in conjunction with the *IBM COBOL Programming Guide* for your platform.

## IBM Extensions

IBM extensions generally add to language element rules or restrictions. These extensions are highlighted in blue ink. The blue highlighting may appear within a sentence, figure, table, or syntax diagram. For example:

x IBM extensions in text are shown this way.

Extensions are not indicated in the table of contents, appendixes, glossary, or index.

## Obsolete Language Elements

Obsolete language elements are COBOL 85 Standard language elements that will be deleted from the next revision of the Standard. (This does **not** imply that these elements will be eliminated from a future release of an IBM COBOL compiler.)

The language elements that will be deleted from the next revision of the COBOL 85 Standard are:

- ALTER statement
- AUTHOR paragraph
- Comment entry


© Copyright IBM Corp. 1991, 1996

- DATA RECORDS clause
- DATE-COMPILED paragraph
- DATE-WRITTEN paragraph
- DEBUG-ITEM special register
- Debugging sections
- ENTER statement
- GO TO without a specified procedure name
- INSTALLATION paragraph
- LABEL RECORDS clause
- MEMORY SIZE clause
- MULTIPLE FILE TAPE clause
- REVERSED phrase
- SECURITY paragraph
- SEGMENT-LIMIT
- SEGMENTATION
- STOP statement
- USE FOR DEBUGGING declarative
- VALUE OF clause
- The figurative constant ALL literal, when associated with a numeric or numeric-edited item and with a length greater than one

## How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►— symbol indicates the beginning of a syntax diagram.

  The —► symbol indicates that the syntax diagram is continued on the next line.

  The ►— symbol indicates that the syntax diagram is continued from the previous line.

  The —►◄ symbol indicates the end of a syntax diagram.

  Diagrams of syntactical units other than complete statements start with the ►— symbol and end with the —► symbol.

- Required items appear on the horizontal line (the main path).

  **Format**

  ```
  ►►—STATEMENT—required item———————————————————————————►◄
  ```

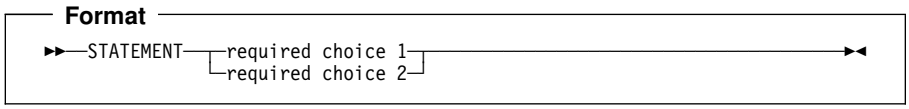
- Optional items appear below the main path.

  **Format**

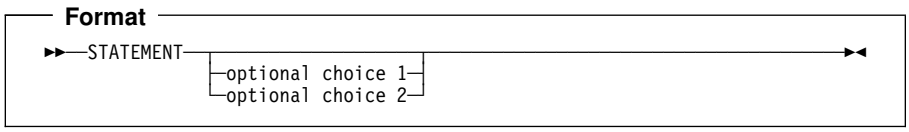
  ```
  ►►—STATEMENT—┬———————————————————————————————————————►◄
               └—optional item—┘
  ```

- When you can choose from two or more items, they appear vertically, in a stack.

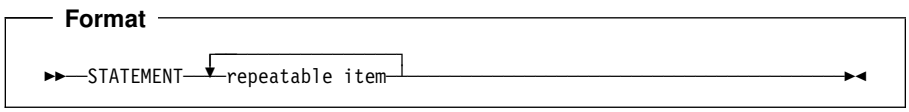
  If you **must** choose one of the items, one item of the stack appears on the main path.




  If choosing one of the items is optional, the entire stack appears below the main path.




- An arrow returning to the left above the main line indicates an item that can be repeated.

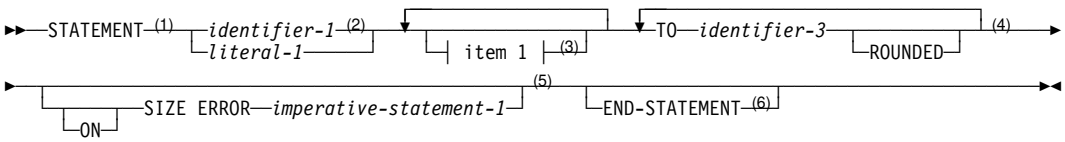



  A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

- Variables appear in all lowercase letters (for example, `parmx`). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, they must be entered as part of the syntax.

The following example shows how the syntax is used.

**Format**




**item 1:**

**Notes:**

1 The STATEMENT key word must be specified and coded as shown.

2 This operand is required. Either identifier-1 or literal-1 must be coded.

3 The item 1 fragment is optional; it can be coded or not, as required by the application. If item 1 is coded, it can be repeated with each entry separated by one or more COBOL separators. Entry selections allowed for this fragment are described at the bottom of the diagram.

4 The operand identifier-3 and associated TO key word are required and can be repeated with one or more COBOL separators separating each entry. Each entry can be assigned the key word ROUNDED.

5 The ON SIZE ERROR phrase with associated imperative-statement-1 are optional. If the ON SIZE ERROR phrase is coded, the key word ON is optional.

6 The END-STATEMENT key word can be coded to end the statement. It is not a required delimiter.

7 For the published manual, blue indicates that arithmetic-expression-1 is an IBM extension. For PostScript files provided with the product and BookManager BOOK files, an 'X' revision bar indicates that arithmetic-expression-1 is an IBM extension. This operand is optional.

## DBCS Notation

MVS/VM Double-Byte Character Strings (DBCS) in literals, comments, and user-defined words are delimited by shift-out and shift-in characters. In this manual, the shift-out delimiter is represented pictorially by the **<** character, and the shift-in character is represented pictorially by the **>** character. The EBCDIC codes for the shift-out and shift-in delimiters are X'0E' and X'0F', respectively.

The **<>** symbol denotes contiguous shift-out and shift-in characters. The **><** symbol denotes contiguous shift-in and shift-out characters.

Double-byte characters are represented in this form: **D1D2D3**.

EBCDIC characters in double-byte form are represented in this form :**.A.B.C**. The dots separating the letters represent the hexadecimal value X'42'.

WRKSTN Under AIX, OS/2, and Windows, you do not delimit DBCS character strings by shift-in or shift-out characters.

## Acknowledgment

The following extract from Government Printing Office Form Number 1965-0795689 is presented for the information and guidance of the user:

> Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention COBOL in acknowledgment of the source, but need not quote this entire section.
>
> COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.
>
> No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection there with.
>
> Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.
>
> The authors and copyright holders of copyrighted material:
>
> FLOW-MATIC (Trademark of Sperry Rand Corporation),
> Programming for the UNIVAC (R) I and II, Data
> Automation Systems copyrighted 1958, 1959, by
> Sperry Rand Corporation; IBM Commercial Translator,
> Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI
> 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell,
>
> have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

# Summary of Changes

This summary of changes lists the major changes for the indicated IBM COBOL product, as well as changes made specifically to this *IBM COBOL Language Reference*.

## September 1996

The following section lists the key changes that have been made to the VisualAge for COBOL for OS/2 product with the Release 2 refresh. Those documented in this publication have an associated page reference for your convenience.

- Support for and CDECL calling convention ("CALLINTERFACE" on page 514)
- Support for alternate index files on the ASSIGN statement ("ASSIGN Clause" on page 103)
- Support for Open Database Connectivity (ODBC) that provides you access to data from a variety of databases and file systems
- Windows First Step provides an IBM COBOL application development environment on Windows 95 and Windows NT that includes a 32-bit compiler, run-time, editor, debug tool, and Performance Analyzer
- Support for the Standard Language File System (STL File System) which supports sequential, indexed, and relative files on the local system efficiently and consistently across workstation and PC systems

## July 1996

The following section lists the key changes that have been made to the IBM VisualAge for COBOL for OS/2 product since Release 1. Those documented in this publication have an associated page reference for you convenience.

Note, this edition of the *COBOL Language Reference* (SC26-4769-01) is available only in softcopy format, as refreshed viewable and printable files for AIX and OS/2 and in BookManager format (on the next available MVS collection kit) for MVS and VM.

- Support for shared install
  - Install only those files that must be local to your hard drive
  - Remainder of the files remain on the CD-ROM or on a LAN server
- Redeveloper tools (Professional Only)
  - Understand the resources, structure, and flow of your applications
  - Understand your program logic, control flow, and data flow
  - Upgrade existing applications by converting and structuring older COBOL programs to newer levels of COBOL
- Transaction Assistant: improved dialogue and steps

© Copyright IBM Corp. 1991, 1996

- Packaging Tool: enhanced diskette-generating interface and reduced space requirements
- Integrated SQL processing
  - Eliminates the need for DB2 pre-processing
  - Removes restrictions such as the use OO extensions, nested programs, and batch compilation with SQL statements.
- Support for `_FAR16_Pascal` calling convention ("CALLINTERFACE" on page 514)
- Optimization of compiler-generated code
- Transparent access of MVS PDS members for COPY files
- Support for source file extensions other than .CBL
- Support for binary collating sequence for enhanced performance

---

## October 1995

The following changes have been made to the IBM COBOL for MVS & VM (formerly named IBM SAA AD/Cycle COBOL/370) language:

### Extensions for Object-Oriented COBOL (MVS, AIX, OS/2, and Windows Only)

The following extensions to the COBOL language enable object-oriented COBOL programming:

- CLASS IDENTIFICATION DIVISION
- CLASS ENVIRONMENT DIVISION
- CLASS DATA DIVISION
- CLASS PROCEDURE DIVISION
- METHOD IDENTIFICATION DIVISION
- METHOD ENVIRONMENT DIVISION
- METHOD DATA DIVISION
- METHOD PROCEDURE DIVISION
- USAGE OBJECT REFERENCE clause
- EXIT statement extension
- INVOKE statement
- SET statement extension

### Extensions for Interoperability (MVS, VM, AIX, and OS/2)

The following extensions to the COBOL language enable improved interoperability between COBOL and C, SOM, and Language Environment:

- CALL ... RETURNING for invocation of functions
- CALL procedure-pointer
- SET procedure-pointer to function pointer
- OMITTED arguments
- Recursion
- Null-terminated literal strings
- BY VALUE arguments on CALL and INVOKE (both identifiers and literals)

- BY VALUE and BY REFERENCE declarations for parameters on the PROCEDURE DIVISION and ENTRY USING statements
- PROCEDURE DIVISION RETURNING phrase for specifying a return value from a program, method, or C function.
- Long and mixed-case program names

## Support for COBOL on AIX and OS/2

For details on language differences, see Appendix G, “Summary of Language Difference: Host COBOL and Workstation COBOL” on page 544.

# Part 1. COBOL Language Structure


**Characters** . . . . 4
Character-Strings . . . . 5
Figurative Constants . . . . 10
Special Registers . . . . 12
Literals . . . . 22
Separators . . . . 30

**Sections and Paragraphs** . . . . 32
Statements and Clauses . . . . 32

**Reference Format** . . . . 34
Sequence Number Area . . . . 34
Indicator Area . . . . 34
Area A . . . . 35
Area B . . . . 37
Area A or Area B . . . . 39

**Scope of Names** . . . . 41
Types of Names . . . . 41
External and Internal Resources . . . . 44
Resolution of Names . . . . 45

**Referencing Data Names, Copy Libraries, and Procedure Division Names** . 46
Uniqueness of Reference . . . . 46

**Transfer of Control** . . . . 58


© Copyright IBM Corp. 1991, 1996

# Characters

The most basic and indivisible unit of the COBOL language is the **character**. The IBM COBOL character set includes the letters of the alphabet, digits, and special characters. The complete set of characters that form the IBM COBOL character set is shown in Table 1.

The basic IBM COBOL language is restricted to the character set shown in Table 1, but the content of nonnumeric literals, comment lines, comment entries, and data can include any of the characters from the character set of the computer.

x In some cases, the basic character set is extended with the national character set. The
x national character set support includes the Double-Byte Character Set (DBCS) and,
x additionally for AIX, the Extended Unix** Code (EUC) code page.

x Double-byte characters, as the name implies, occupy two adjacent bytes to represent 1
x character. A character string containing DBCS characters is called a **DBCS character-**
x **string**.

x AIX For AIX, characters from the EUC code page can be from one to four bytes
x long.

x DBCS and EUC characters are valid characters in certain COBOL character-strings.
x For details, see “COBOL Words with Multi-Byte Characters” on page 6 and “DBCS
x Literals” on page 27.

Individual characters are joined to form **character-strings**, **separators**, and **text words**.

A **character-string** is a character or a sequence of contiguous characters that forms a COBOL word, a literal, a PICTURE character-string, or a comment-entry. A character-string is delimited by separators.

A **separator** is a string of one or two contiguous characters used to delimit character strings. Separators are described in detail under “Separators” on page 30.

A **text word** is a character or a sequence of contiguous characters between character positions 8 and 72 inclusive on a line in a COBOL library, source program, or in pseudo-text. For more information on pseudo-text, see “Pseudo-Text” on page 40.

*Table 1 (Page 1 of 2). Characters—Meanings*

| Character | Meaning |
|---|---|
| b | Space |
| + | Plus sign |
| – | Minus sign or Hyphen |
| * | Asterisk |
| / | Slant, Solidus, Stroke, or Slash |
| = | Equal sign |

*Table 1 (Page 2 of 2). Characters—Meanings*

| Character | Meaning |
|---|---|
| $ | Currency sign |
| , | Comma |
| ; | Semicolon |
| . | Decimal point or Period |
| " | Quotation mark |
| ( | Left parenthesis |
| ) | Right parenthesis |
| > | Greater than |
| < | Less than |
| : | Colon |
| x ' | Apostrophe |
| A–Z | Alphabet (uppercase) |
| a–z | Alphabet (lowercase) |
| 0–9 | Numeric characters |

## Character-Strings

x x You can use EBCDIC and/or DBCS characters strings under MVS and VM or ASCII and/or DBCS/EUC character-strings under AIX, OS/2, and Windows to form the following:

- COBOL words
- Literals
- PICTURE character-strings (EBCDIC or ASCII character-strings only)
- Comment text

## COBOL Words with Single-Byte Characters

A COBOL word is a character-string of not more than 30 characters that forms a user-defined word, a system-name, or a reserved word. Except for arithmetic operators and relation characters, each character of a COBOL word is selected from the following:

- A through Z
- a through z
- 0 through 9
- - (hyphen)

The hyphen cannot appear as the first or last character in such words. All user-defined words (except for section-names, paragraph-names, segment-numbers, and level-numbers) must contain at least one alphabetic character. Segment numbers and level numbers need not be unique; a given specification of a segment-number or level-number can be identical to any other segment-number or level-number. Each lowercase letter is considered to be equivalent to its corresponding uppercase letter, except in nonnumeric literals.

Within a source program the following rules apply for all COBOL words with single-byte characters:

- A reserved word cannot be used as a user-defined word or as a system-name.

- The same COBOL word, however, can be used as both a user-defined word and as a system-name. The classification of a specific occurrence of a COBOL word is determined by the context of the clause or phrase in which it occurs.

## COBOL Words with Multi-Byte Characters

DBCS/EUC characters must conform to the normal COBOL rules for user-defined words. The following are the rules for forming user-defined words from multi-byte characters:

*Table 2. Rules for Forming Words from Multi-Byte Characters*

| Rule | MVS and VM | AIX, OS/2, and Windows |
|---|---|---|
| Use of Shift-Out Shift-In Characters | DBCS user-defined words begin with a shift-out character and end with a shift-in character. | Not required |
| Value Range | DBCS user-defined words can contain characters whose values range from X'41' to X'FE' for both bytes, and X'4040' (DBCS space). In addition, DBCS data items and literals can include characters that range in hexadecimal value from X'00' to X'FF' for both bytes. | Valid value ranges for multi-byte characters depend on the specific code page being used. |
| Containing Characters | DBCS user-defined words can contain only double-byte characters, and must contain at least one non-EBCDIC character. (Double-byte EBCDIC characters are represented by X'42' in the first byte.) Single-byte characters are not allowed in a DBCS word.<br>DBCS user-defined words can contain both double-byte EBCDIC and double-byte non EBCDIC characters. The only double-byte EBCDIC characters allowed are: A - Z, a - z, 0 - 9, and the hyphen (-). The hyphen cannot appear as the first or last character. The rules that apply to EBCDIC user-defined words also apply to DBCS user-defined words. | A user-defined word can consist of both single-byte or multiple-byte (including double-byte) characters. If a character exists in both single-byte and multiple-byte forms, its single-byte and multi-byte representations are not equivalent. |
| Continuation Rules | Words **cannot** be continued across lines. | Words **cannot** be continued across lines. |
| Uppercase / Lowercase Letters | Equivalent | Not equivalent |
| Maximum Length | 14 characters | 15 characters for a DBCS code page<br>For AIX only:<br>7 characters for EUC code page IBM_eucTW<br>10 characters for EUC code pages, IBM_eucJP, IBM_eucKR, and IBM_eucCN |

## User-Defined Words

The following sets of **user-defined words** are supported:

- alphabet-name
- class-name
- condition-name
- data-name
- file-name
- index-name
- level-numbers: 01-49, 66, 77, 88
- library-name
- method-name
- mnemonic-name
- object-oriented class-name
- paragraph-name
- priority-numbers: 00-99
- program-name
- record-name
- section-name
- symbolic-character
- text-name

Multi-byte characters are valid for user-defined words, except for level numbers, priority numbers, object-oriented class names, and method names.

For level-numbers and priority numbers, each word must be a 1-digit or 2-digit integer.

Within a given source program or class definition, but excluding any contained program or method, each user-defined word (except level-numbers and priority-numbers) can belong to only **one** of these sets. Each user-defined word within a set must be unique, except as specified in "Referencing Data Names, Copy Libraries, and Procedure Division Names" on page 46.

The following types of user-defined words can be referenced by statements and entries in that program in which the user-defined word is declared:

- Paragraph-name
- Section-name

The following types of user-defined words can be referenced by any COBOL program, provided that the compiling system supports the associated library or other system, and the entities referenced are known to that system:

- Library-name
- Text-name

## Character-Strings

The following types of names, when they are declared within a Configuration Section, can be referenced by statements and entries either in that program which contains a Configuration Section or in any program contained within that program:

- Alphabet-name
- Class-name
- Condition-name
- Mnemonic-name
- Symbolic-character

The function of each user-defined word is described in the clause or statement in which it appears.

## System-Names

A **system-name** is a character string that has a specific meaning to the system. There are three types of system-names:

- Computer-name
- Language-name
- Implementor-name

There are three types of implementor-names:

- Environment-name
- External class-name
- Assignment-name

The meaning of each system-name is described with the format in which it appears.

MVS/VM Under MVS and VM, the only DBCS character string system-name allowed is computer-name.

WRKSTN Under AIX, OS/2, and Windows, multi-byte characters are allowed for system-name.

## Function-Names

A **function-name** specifies the mechanism provided to determine the value of an intrinsic function. The same word, in a different context, can appear in a program as a user-defined word or a system-name. For a list of function-names and their definitions, see Table 47 on page 432.

## Reserved Words

A **reserved word** is a character-string with a predefined meaning in a COBOL source program. IBM COBOL reserved words are listed in Appendix D, “Reserved Words” on page 532.

Information on selecting an alternate reserved word table can be found in the *IBM COBOL Programming Guide* for your platform.

## Character-Strings

There are six types of reserved words:

- Keywords
- Optional words
- Figurative constants
- Special character words
- Special object identifiers
- Special registers

**Keywords**

Keywords are reserved words that are required within a given clause, entry, or statement. Within each format, such words appear in uppercase on the main path.

**Optional Words**

Optional words are reserved words that can be included in the format of a clause, entry, or statement in order to improve readability. They have no effect on the execution of the program.

**Figurative Constants**

See “Figurative Constants” on page 10.

**Special Character Words**

There are two types of **special characters**, which are only recognized as special characters when represented in single-byte.

- **Arithmetic operators: + - / * ****

  See “Arithmetic Expressions” on page 223.

- **Relational operators**: < > = <= >=

  See “Conditional Expressions” on page 225.

**Special Object Identifiers**

COBOL provides two special object identifiers, SELF and SUPER, used in a method Procedure Division:

**SELF**

A special object identifier you can use in the Procedure Division of a method. SELF refers to the object instance used to invoke the currently-executing method. You can specify SELF only in source program positions that are explicitly listed in the syntax diagrams.

**SUPER**

A special object identifier you can use in the Procedure Division of a method only as the object identifier in an INVOKE statement. When used in this way, SUPER refers to the object instance used to invoke the currently-executing method. The resolution of the method to be invoked ignores any methods declared in the class definition of the currently-executing method and methods defined in any class derived from that class. Thus, the method invoked is inherited from an ancestor class.

**Special Registers**

See “Special Registers” on page 12.

## Figurative Constants

**Figurative constants** are reserved words that name and refer to specific constant values. The reserved words for figurative constants and their meanings are:

**ZERO/ZEROS/ZEROES**
Represents the numeric value zero (0), or one or more occurrences of the nonnumeric character zero (0), depending on context.

When the context cannot be determined, a nonnumeric zero is used.

**SPACE/SPACES**
Represents one or more blanks or spaces. SPACE is treated as a nonnumeric literal.

**HIGH-VALUE/HIGH-VALUES**
Represents one or more occurrences of the character that has the highest ordinal position in the collating sequence used. For the EBCDIC collating sequence, the character is X'FF'; for other collating sequences, the actual character used depends on the collating sequence indicated by the locale. For more information on locale, see Appendix F, “Locale Considerations (Workstation Only)” on page 543. HIGH-VALUE is treated as a nonnumeric literal.

**LOW-VALUE/LOW-VALUES**
Represents one or more occurrences of the character that has the lowest ordinal position in the collating sequence used. For the EBCDIC collating sequence, the character is X'00'; for other collating sequences, the actual character used depends on the collating sequence. LOW-VALUE is treated as a nonnumeric literal.

**QUOTE/QUOTES**
Represents one or more occurrences of the quotation mark character ("). QUOTE or QUOTES cannot be used in place of a quotation mark to enclose a nonnumeric literal.

Represents one or more occurrences of a nonnumeric literal delimiter depending on the QUOTE compiler option.

x Represents one or more occurrences of a nonnumeric literal delimiter depending
x on the APOST compiler option.

**ALL literal**
Represents one or more occurrences of the string of characters composing the literal. The literal must be either a nonnumeric literal or a figurative constant other than the ALL literal. When a figurative constant, other than the ALL literal is used, the word ALL is redundant and is used for readability only. The figurative constant ALL literal must not be used with the CALL, INSPECT, INVOKE, STOP, or STRING statements.

**symbolic-character**
Represents one or more of the characters specified as a value of the symbolic-character in the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph.

x WRKSTN Under AIX, OS/2, and Windows, you cannot specify the SYMBOLIC
x CHARACTER clause if a DBCS or EUC code page is indicated by the locale
x setting. For more information on locale, see Appendix F, "Locale Considerations
x (Workstation Only)" on page 543.

x **NULL/NULLS**
x Represents a value used to indicate that data items defined with USAGE IS
x POINTER, USAGE IS PROCEDURE-POINTER, USAGE IS OBJECT REFERENCE
x or the ADDRESS OF special register do not contain a valid address. NULL can be
x used only where explicitly allowed in the syntax format. NULL has the value of
x zero.

The singular and plural forms of ZERO, SPACE, HIGH-VALUE, LOW-VALUE, and QUOTE can be used interchangeably. For example, if data-name-1 is a 5-character data item, each of the following statements will fill data-name-1 with five spaces:

```
MOVE SPACE       TO  DATA-NAME-1
MOVE SPACES      TO  DATA-NAME-1
MOVE ALL SPACES  TO  DATA-NAME-1
```

You can use a figurative constant wherever "literal" appears in a syntax diagram, except where explicitly prohibited. When a numeric literal appears in a syntax diagram, only the figurative constant ZERO (ZEROS, ZEROES) can be used. Figurative constants are not allowed as function arguments except in an arithmetic expression, where they are arguments to a function.

The length of a figurative constant depends on the context of the program. The following rules apply:

- When a figurative constant is specified in a VALUE clause or associated with a data item (for example, when it is moved to or compared with another item), the length of the figurative constant character-string is equal to 1 or the number of character positions in the associated data item, whichever is greater.
- When a figurative constant, other than the ALL literal, is not associated with another data item (for example, in a CALL, INVOKE, STOP, STRING, or UNSTRING statement), the length of the character-string is 1 character.

## Special Registers

**Special registers** are reserved words that name storage areas generated by the compiler. Their primary use is to store information produced through specific COBOL features. Each such storage area has a fixed name, and must not be defined within the program.

Unless otherwise explicitly restricted, a special register can be used wherever a dataname or identifier having the same definition as the implicit definition of the special register, (which is specified later in this section).

If qualification is allowed, special registers can be qualified as necessary to provide uniqueness. (For more information, see "Qualification" on page 46.)

x For the first CALL to a program or INVOKE of a method, the compiler initializes the special register fields to their initial values.

In the following cases:

- For subsequent CALLs to a CANCELed program
- Programs that possess the INITIAL attribute
- x • Programs that possess the RECURSIVE attribute
- x • Programs compiled with the THREAD option (Workstation only)

The following special registers are reset to their initial value on each program or method entry:

- x • ADDRESS OF (for each record in the Linkage Section)
- x • RETURN-CODE
- x • SORT-CONTROL
- x • SORT-CORE-SIZE
- x • SORT-FILE-SIZE
- x • SORT-MESSAGE
- x • SORT-MODE-SIZE
- x • SORT-RETURN
- x • TALLY

In all other cases, the special registers will not be reset; (they will be unchanged from x the value contained on the previous CALL or INVOKE.)

You can specify an alphanumeric special register in a function wherever an alphanumeric argument to a function is allowed, unless specifically prohibited.

## ADDRESS OF

The ADDRESS OF special register exists for each record (01 or 77) in the Linkage Section, except for those records that redefine each other. In such cases, the ADDRESS OF special register is similarly redefined.

The ADDRESS OF special register is implicitly defined USAGE IS POINTER.

You can specify the ADDRESS OF special register as an argument to the LENGTH function. If the ADDRESS OF special register is used as the argument to the LENGTH function, the result will always be 4, independent of the argument specified for ADDRESS OF.

A function-identifier is not allowed as the operand of the ADDRESS OF special register.

## DEBUG-ITEM

The DEBUG-ITEM special register provides information for a debugging declarative procedure about the conditions causing debugging section execution.

DEBUG-ITEM has the following implicit description:

```
01  DEBUG-ITEM.
  02   DEBUG-LINE     PICTURE IS X(6).
  02   FILLER         PICTURE IS X  VALUE SPACE.
  02   DEBUG-NAME     PICTURE IS X(30).
  02   FILLER         PICTURE IS X  VALUE SPACE.
  02   DEBUG-SUB-1    PICTURE IS S9999  SIGN IS LEADING SEPARATE CHARACTER.
  02   FILLER         PICTURE IS X  VALUE SPACE.
  02   DEBUG-SUB-2    PICTURE IS S9999  SIGN IS LEADING SEPARATE CHARACTER.
  02   FILLER         PICTURE IS X  VALUE SPACE.
  02   DEBUG-SUB-3    PICTURE IS S9999  SIGN IS LEADING SEPARATE CHARACTER.
  02   FILLER         PICTURE IS X  VALUE SPACE.
  02   DEBUG-CONTENTS PICTURE IS X(n).
```

Before each debugging section is executed, DEBUG-ITEM is filled with spaces. The contents of the DEBUG-ITEM subfields are updated according to the rules for the MOVE statement, with one exception: DEBUG-CONTENTS is updated as if the move were an alphanumeric-to-alphanumeric elementary move without conversion of data from one form of internal representation to another.

After updating, each field contains:

**DEBUG-LINE**
The source-statement sequence number (or the compiler-generated sequence number, depending on the compiler option chosen) that caused execution of the debugging section.

**DEBUG-NAME**
The first 30 characters of the name that caused execution of the debugging section. Any qualifiers are separated by the word “OF.”

## Special Registers

**DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3**

If the DEBUG-NAME is subscripted or indexed, the occurrence number of each level is entered in the respective DEBUG-SUB-n. If the item is **not** subscripted or indexed, these fields remain as spaces. You must not reference the DEBUG-ITEM special register if your program uses more than three levels of subscripting or indexing.

**DEBUG-CONTENTS**

Data is moved into DEBUG-CONTENTS, as shown in Table 3.

*Table 3. DEBUG-ITEM Subfield Contents*

| Cause of Debugging Section Execution | Statement Referred to in DEBUG-LINE | Contents of DEBUG-NAME | Contents of DEBUG-CONTENTS |
|---|---|---|---|
| procedure-name-1 ALTER reference | ALTER statement | procedure-name-1 | procedure-name-n in TO PROCEED TO phrase |
| GO TO procedure-name-n | GO TO statement | procedure-name-n | spaces |
| procedure-name-n in SORT/MERGE input/output procedure | SORT/MERGE statement | procedure-name-n | “SORT INPUT” “SORT OUTPUT” “MERGE OUTPUT” (as applicable) |
| PERFORM statement transfer of control | This PERFORM statement | procedure-name-n | “PERFORM LOOP” |
| procedure-name-n in a USE procedure | Statement causing USE procedure execution | procedure-name-n | “USE PROCEDURE” |
| Implicit transfer from previous sequential procedure | Previous statement executed in previous sequential procedure * | procedure-name-n | “FALL THROUGH” |
| First execution of first nondeclarative procedure | Line number of first nondeclarative procedure-name | first nondeclarative procedure | “START PROGRAM” |

**Note:**

* If this procedure is preceded by a section header, and control is passed through the section header, the statement number refers to the section header.

## LENGTH OF

The LENGTH OF special register contains the number of bytes used by an identifier.

LENGTH OF creates an implicit special register whose content is equal to the current byte length of the data item referenced by the identifier.

**Note:** For DBCS data items, each character occupies 2 bytes of storage.

LENGTH OF can be used in the Procedure Division anywhere a numeric data item having the same definition as the implied definition of the LENGTH OF special register is used. The LENGTH OF special register has the implicit definition:

```
USAGE IS BINARY PICTURE 9(9)
```

If the data item referenced by the identifier contains the GLOBAL clause, the LENGTH OF special register is a global data item.

The LENGTH OF special register can appear within either the starting character position or the length expressions of a reference modification specification. However, the LENGTH OF special register cannot be applied to any operand that is reference-modified.

The LENGTH OF operand cannot be a function, but the LENGTH OF special register is allowed in a function where an integer argument is allowed.

If the LENGTH OF special register is used as the argument to the LENGTH function, the result is always 4, independent of the argument specified for LENGTH OF.

LENGTH OF can **not** be either of the following:

- A receiving data item
- A subscript

When the LENGTH OF special register is used as a parameter in a CALL statement, the parameter must be a BY CONTENT parameter.

When a table element is specified, the LENGTH OF special register contains the length, in bytes, of one occurrence. When referring to a table element, it need not be subscripted.

A value is returned for any identifier whose length can be determined, even if the area referenced by the identifier is currently not available to the program.

A separate LENGTH OF special register exists for each identifier referenced with the LENGTH OF phrase, for example:

```
MOVE LENGTH OF A TO B
DISPLAY LENGTH OF A, A
ADD LENGTH OF A TO B
CALL "PROGX" USING BY REFERENCE A BY CONTENT LENGTH OF A
```

x **Note:** The number of bytes occupied by a COBOL item is also accessible through the
x intrinsic function LENGTH (See “LENGTH” on page 450). LENGTH supports nonnu-
x meric literals in addition to data names.

## LINAGE-COUNTER

A separate LINAGE-COUNTER special register is generated for each FD entry containing a LINAGE clause. When more than one is generated, you must qualify each reference to a LINAGE-COUNTER with its related file-name.

The implicit description of the LINAGE-COUNTER special register is one of the following:

- If the LINAGE clause specifies a data-name, LINAGE-COUNTER has the same PICTURE and USAGE as that data-name.
- If the LINAGE clause specifies an integer, LINAGE-COUNTER is a binary item with the same number of digits as that integer.

For more information, see “LINAGE Clause” on page 151.

The value in LINAGE-COUNTER at any given time is the line number at which the device is positioned within the current page. LINAGE-COUNTER can be referred to in Procedure Division statements; it must not be modified by them.

LINAGE-COUNTER is initialized to 1 when an OPEN statement for its associated file is executed.

LINAGE-COUNTER is automatically modified by any WRITE statement for this file. (See “WRITE Statement” on page 415.)

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, the LINAGE-COUNTER data item is a global data item.

You can specify the LINAGE-COUNTER special register wherever an integer argument to a function is allowed.

## RETURN-CODE

The RETURN-CODE special register can be used to pass information between separately-compiled programs.

You can set the RETURN-CODE special register to pass a return code to the program or the system before executing a STOP RUN statement. It has the implicit definition:

```
01  RETURN-CODE GLOBAL PICTURE S9(4) USAGE BINARY VALUE ZERO
```

The following are examples of how to set the RETURN-CODE special register:

```
COMPUTE RETURN-CODE = 8
```

or

```
MOVE 8 to RETURN-CODE.
```

When control returns to the operating system, the special register contents are returned as a user return code. When used in nested programs, this special register is implicitly defined in the outermost program.

**Note:** The RETURN-CODE special register does not return a value from an invoked method or from a program that uses CALL...RETURNING. For more information, see "INVOKE Statement" on page 319 or "CALL Statement" on page 270.

You can specify the RETURN-CODE special register in a function wherever an integer argument is allowed.

AIX Under AIX only, when your program ends in an abend, the return code is set to 128 plus the signal number.

MVS/VM The RETURN-CODE special register will not contain return code information from a service call for a Language Environment Callable Service . For more information, see the *IBM COBOL for MVS & VM Programming Guide* and *Language Environment Programming Guide*.

## SHIFT-OUT and SHIFT-IN

WRKSTN The SHIFT-OUT and SHIFT-IN special registers are supported; however, the code pages for AIX, OS/2, and Windows do not recognize them as delimiters for double-byte characters.

The SHIFT-OUT and SHIFT-IN special registers are implicitly defined as alphanumeric data items of the format:

```
01    SHIFT-OUT GLOBAL PICTURE X(1) USAGE DISPLAY VALUE X"0E"
01    SHIFT-IN GLOBAL PICTURE X(1) USAGE DISPLAY VALUE X"0F"
```

These special registers represent shift-out and shift-in control characters without the use of unprintable characters.

You can specify the SHIFT-OUT and SHIFT-IN special registers in a function wherever an alphanumeric argument is allowed.

These special registers cannot be receiving items. SHIFT-OUT and SHIFT-IN cannot be used in place of the keyboard control characters when defining DBCS user-defined words and when specifying DBCS literals.

Following is an example of how SHIFT-OUT and SHIFT-IN might be used:

```
DATA DIVISION.
WORKING-STORAGE.

01   DBCSGRP.
       05  SO       PIC X.
       05  DBCSITEM PIC G(3) USAGE DISPLAY-1
       05  SI       PIC X.
                  .
                  .
                  .
PROCEDURE DIVISION.

       MOVE SHIFT-OUT TO SO
       MOVE G"<D1D2D3>" TO DBCSITEM
       MOVE SHIFT-IN TO SI
       DISPLAY DBCSGRP
```

When used in nested programs, this special register is implicitly defined in the outermost program.

## SORT-CONTROL

The SORT-CONTROL special register is the name of an alphanumeric data item.

WRKSTN Under AIX, OS/2, and Windows, it is implicitly defined as:

```
01 SORT-CONTROL GLOBAL PICTURE X(160) VALUE "file name".
```

Where "file name" is the file name used by SmartSORT as the source for additional sort/merge options.

MVS/VM Under MVS and VM it is implicitly defined as:

```
01    SORT-CONTROL GLOBAL PICTURE X(8) USAGE DISPLAY VALUE "IGZSRTCD"
```

This register contains the ddname of the data set that holds the control statements used to improve the performance of a sorting or merging operation.

Under MVS, you can provide a DD statement for the data set identified by the SORT-CONTROL special register, and IBM COBOL for MVS & VM will attempt to open the data set at execution time. Any error will be diagnosed with an informational message.

You can specify the SORT-CONTROL special register in a function wherever an alphanumeric argument is allowed.

The SORT-CONTROL special register is not necessary for a successful sorting or merging operation.

Note that the sort control file takes precedence over the SORT special registers.

When used in nested programs, this special register is implicitly defined in the outermost program.

For further information, see the *IBM COBOL Programming Guide* for your platform.

## SORT-CORE-SIZE

The SORT-CORE-SIZE special register is the name of a binary data item that you can use to specify the number of bytes of storage available to the sort utility. It has the implicit definition:

```
01    SORT-CORE-SIZE GLOBAL PICTURE S9(8) USAGE BINARY VALUE ZERO
```

WRKSTN Under AIX, OS/2, and Windows, the amount of storage indicated in the SORT-CORE-SIZE special register does not include memory areas required by COBOL library functions not related to the SORT or MERGE function. It also does not include fixed amount of memory areas (modules, control blocks, fixed size work areas) required for the sort and merge implementation.

MVS/VM Under MVS and CMS, SORT-CORE-SIZE can be used in place of the MAINSIZE or RESINV control statements in the sort control file.

The 'MAINSIZE=' option control statement key word is equivalent to SORT-CORE-SIZE with a positive value.

The 'RESINV=' option control statement key word is equivalent to SORT-CORE-SIZE with a negative value.

The 'MAINSIZE=MAX' option control statement key word is equivalent to SORT-CORE-SIZE with a value of +999999 or +99999999.

You can specify the SORT-CORE-SIZE special register in a function wherever an integer argument is allowed.

When used in nested programs, this special register is implicitly defined in the outermost program.

## SORT-FILE-SIZE

The SORT-FILE-SIZE special register is the name of a binary data item that you can use to specify the estimated number of records in the sort input file, file-name-1. It has the implicit definition:

```
01     SORT-FILE-SIZE GLOBAL PICTURE S9(8) USAGE BINARY VALUE ZERO
```

WRKSTN Under AIX, OS/2, and Windows, references to the SORT-FILE-SIZE special register are resolved by the compiler; however, the value in the special register has no impact for the execution of a SORT or MERGE statement.

MVS/VM Under MVS and CMS, SORT-FILE-SIZE is equivalent to the 'FILSZ=Ennn' control statement in the sort control file.

You can specify the SORT-FILE-SIZE special register in a function wherever an integer argument is allowed.

When used in nested programs, this special register is implicitly defined in the outermost program.

## SORT-MESSAGE

The SORT-MESSAGE special register is the name of an alphanumeric data item that is available to both sort and merge programs.

WRKSTN Under AIX, OS/2, and Windows, references to the SORT-MESSAGE special register are resolved by the compiler; however, the value in the special register has no impact for the execution of a SORT or MERGE statement.

MVS/VM Under MVS and CMS, it has the implicit definition:

```
01    SORT-MESSAGE GLOBAL PICTURE X(8) USAGE DISPLAY VALUE "SYSOUT"
```

You can use the SORT-MESSAGE special register to specify the ddname of a data set that the sort utility should use in place of the SYSOUT data set.

The ddname specified in SORT-MESSAGE is equivalent to the name specified on the 'MSGDDN=' control statement in the sort control file.

You can specify the SORT-MESSAGE special register in a function wherever an alphanumeric argument is allowed.

When used in nested programs, this special register is implicitly defined in the outermost program.

## SORT-MODE-SIZE

The SORT-MODE-SIZE special register is the name of a binary data item that you can use to specify the length of variable-length records that occur most frequently. It has the implicit definition:

```
01     SORT-MODE-SIZE GLOBAL PICTURE S9(5) USAGE BINARY VALUE ZERO
```

WRKSTN Under AIX, OS/2, and Windows, references to the SORT-MODE-SIZE special register are resolved by the compiler; however, the value in the special register has no impact for the execution of a SORT or MERGE statement.

MVS/VM SORT-MODE-SIZE is equivalent to the 'SMS=' control statement in the sort control file.

You can specify the SORT-MODE-SIZE special register in a function wherever an integer argument is allowed.

When used in nested programs, this special register is implicitly defined in the outermost program.

## SORT-RETURN

The SORT-RETURN special register is the name of a binary data item and is available to both sort and merge programs.

The SORT-RETURN special register has the implicit definition:

```
01     SORT-RETURN GLOBAL PICTURE S9(4) USAGE BINARY VALUE ZERO
```

It contains a return code of 0 (successful) or 16 (unsuccessful) at the completion of a sort/merge operation. If the sort/merge is unsuccessful and there is no reference to this special register anywhere in the program, a message is displayed on the terminal.

You can set the SORT-RETURN special register to 16 in an error declarative or input/output procedure to terminate a sort/merge operation before all records are processed. The operation is terminated on the next input or output function for the SORT or MERGE operation.

You can specify the SORT-RETURN special register in a function wherever an integer argument is allowed.

When used in nested programs, this special register is implicitly defined in the outermost program.

x ### TALLY

x The TALLY special register is the name of a binary data item with the following defi-
x nition:

x ```
01    TALLY GLOBAL PICTURE 9(5) USAGE BINARY VALUE ZERO
```

x You can refer to or modify the contents of TALLY.

x You can specify the TALLY special register in a function wherever an integer argument
x is allowed.

x When used in nested programs, this special register is implicitly defined in the outer-
x most program.

x ### WHEN-COMPILED

x The WHEN-COMPILED special register contains the date at the start of the compila-
x tion. WHEN-COMPILED is an alphanumeric data item with the implicit definition:

x ```
01    WHEN-COMPILED GLOBAL PICTURE X(16) USAGE DISPLAY
```

x The WHEN-COMPILED special register has the format:

x ```
MM/DD/YYhh.mm.ss (MONTH/DAY/YEARhour.minute.second)
```

x For example, if compilation began at 2:04 PM on 27 April 1995, WHEN-COMPILED
x would contain the value 04/27/9514.04.00.

x WHEN-COMPILED can only be used as the sending field in a MOVE statement.

x WHEN-COMPILED special register data cannot be reference-modified.

x When used in nested programs, this special register is implicitly defined in the outer-
x most program.

x **Note:** The compilation date and time is also accessible via the date/time intrinsic func-
x tion WHEN-COMPILED (See “WHEN-COMPILED” on page 478). That function sup-
x ports 4-digit year values, and provides additional information.

---

## Literals

A **literal** is a character-string whose value is specified either by the characters of which it is composed, or by the use of a figurative constant. (See “Figurative Constants” on page 10.) The literal types are **nonnumeric** and **numeric**.

x DBCS literals are also literal types.

### Nonnumeric Literals

A **nonnumeric literal** is a character string enclosed in quotation marks ("), and can contain any allowable character from the character set of the computer. The maximum length of a nonnumeric literal is 160 characters.
```

The enclosing quotation marks are excluded from the literal when the program is compiled. An embedded quotation mark must be represented by a pair of quotation marks (""). For example,

```
"THIS ISN""T WRONG"
```

x As an IBM extension, you can use apostrophes as the literal delimiters instead of
x quotes (independent of the APOST/QUOTE compiler option). An embedded apos-
x trophe must be represented by a pair of apostrophes (''). For example,

```
x 'THIS ISN''T WRONG'
```

The delimiter character used as the opening delimiter for a literal must be used as the closing delimiter for that literal. For example,

```
x 'THIS IS RIGHT'
"THIS IS RIGHT"
'THIS IS WRONG"
```

Any punctuation characters included within a nonnumeric literal are part of the value of the literal.

Every nonnumeric literal is in the **alphanumeric** data category. (Data categories are described in "Classes and Categories of Data" on page 136.)

x Table 4 lists when nonnumeric literals with double-byte or multiple-byte characters
x cannot be used.

x *Table 4. When Multi-Byte Characters are not Allowed in Nonnumeric Literals*

| x **MVS and VM** | **AIX, OS/2, and Windows** |
|---|---|
| x As a literal in the following: | As a literal in the following: |
| x ALPHABET clause | ALPHABET clause |
| x ASSIGN clause | ASSIGN clause |
| x CALL statement program-id | CLASS clause |
| x CANCEL statement | CURRENCY SIGN clause |
| x CLASS clause | END METHOD header |
| x CURRENCY SIGN clause | METHOD-ID paragraph |
| x END METHOD header | PADDING CHARACTER clause |
| x END PROGRAM header | RERUN clause |
| x ENTRY statement | STOP statement |
| x METHOD-ID paragraph | |
| x PADDING CHARACTER clause | |
| x PROGRAM-ID paragraph | |
| x RERUN clause | |
| x STOP statement | |
| x BASIS statement (basis-name) | |
| x COPY statement (text-name) | |
| x COPY statement (library-name) | |

## Literals

WRKSTN Under AIX, OS/2, and Windows, you can include multi-byte as well as single-byte DBCS or EUC characters as alphanumeric literals (such as to initialize display fields). However, COBOL semantics applied to literals that contain both multi-byte and single-byte characters are not sensitive to the length (in bytes) of the individual characters.

The rule of formation for mixed literals are as follows:

- A nonnumeric literal (whether it contains any multi-byte characters or not) is delimited by either an opening and closing " or an opening and closing '. The " or ' must be represented as a single-byte character.
- Nonnumeric literals containing a multi-byte character cannot be continued. The maximum length of a nonnumeric literal with multi-byte characters is limited only by the available positions in Area B on a single source line.

MVS/VM Under MVS and VM, with the DBCS compiler option, the characters X'0E' and X'0F' in a nonnumeric literal will be recognized as shift codes for DBCS characters. That is, the characters between paired shift codes will be recognized as DBCS characters. Unlike a nonnumeric literal compiled under the NODBCS option, additional syntax rules apply to DBCS characters in a nonnumeric literal.

These nonnumeric literals with double-byte characters have the following format:

**Nonnumeric Literals with Double-Byte Characters**

```
"EBCDIC–data<D1D2>EBCDIC–data"
```

**"** The opening and closing delimiter (Alternatively, you can use apostrophes (') as delimiters.)

**<** Represents the shift-out control character (X'0E')

**>** Represents the shift-in control character (X'0F')

Shift-out and shift-in control characters are part of the literal and must be paired with zero or an even number of intervening bytes.

Nested shift codes are not allowed in the DBCS portion of the literal.

The syntax rules for EBCDIC parts of the literal follow the rules for nonnumeric literals. The syntax rules for DBCS parts of the literal follow the rules for DBCS literals. The move and comparison rules for nonnumeric literals with double-byte characters are the same as those for any nonnumeric literal.

The length of a nonnumeric literal with double-byte characters is its byte length, including the shift control characters. The maximum length is limited by the available space on one line in Area B. A nonnumeric literal with double-byte characters cannot be continued.

A nonnumeric literal with double-byte characters is of the alphanumeric category.

## Literals

x Under IBM COBOL for MVS & VM, COBOL statements process nonnumeric literals
x with double-byte characters without sensitivity to the shift codes and character codes.
x The use of statements that operate on a byte-to-byte basis (for example, STRING and
x UNSTRING) can result in strings that are not valid mixtures of EBCDIC and double-byte
x characters. You must be certain that the statements use DBCS characters. See *IBM*
x *COBOL for MVS & VM Programming Guide* for more information on using nonnumeric
x literals and data items with double-byte characters in statements that operate on a byte-
x by-byte basis.

x Hexadecimal notation can be used for nonnumeric literals. This hexadecimal notation
x has the following format:

x **Hexadecimal Notation Format for Nonnumeric Literals**

x
```
X"hexadecimal-digits"
```

x **X"** The opening delimiter for hexadecimal notation of a nonnumeric literal. (Alterna-
x tively, you can use apostrophes (') as delimiters.)

x **"** The closing delimiter for the hexadecimal notation of a nonnumeric literal. (Alterna-
x tively, you can use apostrophes (') as delimiters.)

x Hexadecimal digits can be characters in the range '0' to '9', 'a' to 'f', and 'A' to
x 'F', inclusive. Two hexadecimal digits represent a single character in the
x EBCDIC/ASCII character set. An even number of hexadecimal digits must be specified.
x The maximum length of a hexadecimal literal is 320 hexadecimal digits.

x The continuation rules are the same as those for any nonnumeric literal. The opening
x delimiter (X" or X') cannot be split across lines.

x The DBCS compiler option has no effect on the processing of hexadecimal notation of
x nonnumeric literals.

x The compiler will convert the hexadecimal literal into a normal nonnumeric literal.
x Hexadecimal notation for nonnumeric literals can be used anywhere nonnumeric literals
x can appear.

x The padding character for hexadecimal notation of nonnumeric literals is the blank
x (X'40' for MVS and VM) or (X'20' for AIX, OS/2, and Windows).

## Literals

x Nonnumeric literals can be null-terminated, with the following format:

x **Format for Null-Terminated Nonnumeric Literals**

x
```
Z"ddddd"
```

x **Z"** The opening delimiter for null-terminated notation of a nonnumeric literal. (Alterna-
x tively, you can use apostrophes (') as delimiters.)

x **"** The closing delimiter for a null-terminated notation of a nonnumeric literal. (Alter-
x natively, you can use apostrophes (') as delimiters.)

x Null-terminated nonnumeric literals can be from 0 to 159 characters. You can specify
x any character except X'00', which is the null string automatically appended to the end
x of the literal. The length of the literal includes the terminating null character.

x Null-terminated literals can be used anywhere a nonnumeric literal can be specified and
x have the normal semantics of nonnumeric literals.

x Both characters of the opening delimiter for null-terminated literals (Z" or Z') must be on
x the same source line.

x The LENGTH intrinsic function, when applied to a null-terminated literal, returns the
x number of characters in the literal prior to but not including the terminating null. (The
x LENGTH special register does not support literal operands.)

x Null-terminated literals are not supported in “ALL literal” constructions.

## Numeric Literals

A **numeric literal** is a character-string whose characters are selected from the digits 0 through 9, a sign character (+ or -), and the decimal point. If the literal contains no decimal point, it is an integer. (In this manual, the word **integer** appearing in a format represents a numeric literal of nonzero value that contains no sign and no decimal point; any other restrictions are included with the description of the format.) The following rules apply:

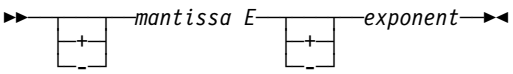
- One through 18 digits are allowed.
- Only one sign character is allowed. If included, it must be the leftmost character of the literal. If the literal is unsigned, it is a positive value.
- Only one decimal point is allowed. If a decimal point is included, it is treated as an assumed decimal point (that is, as not taking up a character position in the literal). The decimal point can appear anywhere within the literal except as the rightmost character.

The value of a numeric literal is the algebraic quantity expressed by the characters in the literal. The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user.

x Numeric literals can be either fixed-point or floating-point numbers.

### Rules for Floating-point Literal Values:

- A floating-point literal is written in the form:




- The sign is optional before the mantissa and the exponent; if you omit the sign, the compiler assumes a positive number.
- The mantissa can contain between 1 and 16 digits. A decimal point must be included in the mantissa.
- The exponent is represented by an E followed by an optional sign and one or two digits.
- MVS/VM Under MVS and VM, the magnitude of a floating-point literal value must fall between 0.54E-78 and 0.72E+76. For values outside of this range, an E-level diagnostic will be produced and the value will be replaced by either 0 or 0.72E+76, respectively.
- WRKSTN Under AIX, OS/2, and Windows, the magnitude of a floating-point literal value must fall between:
  - 32-bit representation—$1.175(10^{-38})$ to $3.403(10^{38})$
  - 64-bit representation—$2.225(10^{-308})$ to $1.798(10^{308})$

Every numeric literal is in the numeric data category. (Data categories are described under “Classes and Categories of Data” on page 136.)

## DBCS Literals

Table 5 lists the formats and rules for DBCS literals. You can use either quotes or apostrophes for the opening and closing delimiters.

*Table 5 (Page 1 of 2). Format and Rules for Forming DBCS Literals*

| Rules | MVS and VM | AIX, OS/2, and Windows |
|---|---|---|
| Format | G"<D1D2D3>"<br>N"<D1D2D3>" | G"D1D2D3"<br>N"D1D2D3" |
| G" N" | Opening delimiters. They must be followed immediately by a shift-out control character.<br><br>For N-literals, when embedded quotes/apostrophes are specified as part of DBCS characters in a DBCS literal, a single embedded DBCS quote/apostrophe is represented by 2 DBCS quotes/apostrophes. If a single embedded DBCS quote/apostrophe is found, an E-level compiler message will be issued and a second embedded DBCS quote/apostrophe will be assumed. | Opening delimiters. |

*Table 5 (Page 2 of 2). Format and Rules for Forming DBCS Literals*

| Rules | MVS and VM | AIX, OS/2, and Windows |
|---|---|---|
| < | Represents the shift-out control character (X'0E') | N/A |
| > | Represents the shift-in control character (X'0F') | N/A |
| " | The closing delimiter. They must appear immediately after the shift-in control character.<br><br>Single-byte quotation marks or apostrophes can appear as part of DBCS characters in a DBCS literal between the shift-out and shift-in control characters. | The closing delimiter. |
| Character Range | X'00' to X'FF' for both bytes, except for X'0F7F' (or X'0F7D' if using apostrophes as the opening and closing delimiters). | Any double-byte character in a DBCS code page. |
| Maximum Length | 28 Characters | N/A |
| Continuation Rules | Cannot be continued across lines. | Cannot be continued across lines. |

## When DBCS Literals are Allowed

DBCS literals are allowed in the following:

- Data Division
  - In the VALUE clause of DBCS data description entries. If you specify a DBCS literal in a VALUE clause for a data item, the length of the literal must not exceed the size indicated by the data item's PICTURE clause. Explicitly or implicitly defining a DBCS data item as USAGE DISPLAY-1 specifies that the data item is to be stored in character form, one character to each 2 bytes.
  - In the VALUE OF clause of file description entries.
- Procedure Division
  - As the sending item when a DBCS or group item is the receiving item.
  - In a relation condition when the comparand is a DBCS or group item.
  - As the figurative constants SPACE/SPACES, ALL SPACE/SPACES, or ALL DBCS literal. These are the only figurative constants that can be DBCS literals.

### When DBCS Literals are Not Allowed

DBCS literals are **not** allowed in the following:

- Non-Procedure Division
  - ALPHABET clause
  - ASSIGN clause
  - CLASS clause
  - CURRENCY SIGN clause
  - END METHOD header
  - END PROGRAM header
  - METHOD-ID paragraph
  - PADDING CHARACTER clause
  - PROGRAM-ID paragraph
  - RERUN clause
- Procedure Division
  - CALL statement (program-name)
  - CANCEL statement
  - ENTRY statement
  - INVOKE statement
  - SET procedure-pointer to ENTRY literal
  - STOP statement
- As a file assignment name
- As a function argument
- As a basis-name in a BASIS statement
- As a text-name or library-name in a COPY statement

## PICTURE Character-Strings

A **PICTURE character-string** is composed of the currency symbol and certain combinations of characters in the COBOL character set. PICTURE character-strings are delimited only by the separator space, separator comma, separator semicolon, or separator period.

A chart of PICTURE clause symbols appears in Table 12 on page 171.

## Comments

A **comment** is a character-string that can contain any combination of characters from the character set of the computer. It has no effect on the execution of the program. There are two forms of comments:

**Comment entry (Identification Division)**
This form is described under “Optional Paragraphs” on page 81.

**Comment line (Any division)**
This form is described under “Comment Lines” on page 39.

## Separators

x Character-strings that form comments can contain:

x • MVS/VM Under MVS and VM, DBCS characters or a combination of DBCS and
x EBCDIC characters.

x • WRKSTN Under AIX, OS/2, and Windows any character from the code page in
x effect.

x Multiple comment lines containing DBCS/EUC strings are allowed. The embedding of
x DBCS/EUC characters in a comment line must be done on a line-by-line basis.
x DBCS/EUC words cannot be continued to a following line. No syntax checking for valid
x DBCS/EUC strings is provided in comment lines.

## Separators

A **separator** is a string of one or more contiguous characters as shown in Table 6.

*Table 6. Separator Characters*

| Separator | Meaning |
|---|---|
| ␣ | Space |
| ,␣ | Comma |
| .␣ | Period |
| ;␣ | Semicolon |
| ( | Left parenthesis |
| ) | Right parenthesis |
| : | Colon |
| "␣ | Quotation marks |
| x '␣ | Apostrophe |
| x X" | Opening delimiter for a nonnumeric literal |
| x Z" | Opening delimiter for a null-terminated nonnumeric literal |
| x N" | Opening delimiter for a DBCS literal |
| x G" | Opening delimiter for a DBCS literal |
| == | Pseudo-text delimiter |

## Rules for Separators

In the following description, {} enclose each separator. Anywhere a space is used as a separator, or as part of a separator, more than one space can be used.

**Space {␣}**

A space can immediately precede or follow any separator except:

- The opening pseudo-text delimiter, where the preceding space is required.
- Within quotation marks. Spaces between quotation marks are considered part of the nonnumeric literal; they are not considered separators.

**Period {.␣}, Comma {,␣}, Semicolon {;␣}**

A separator comma is composed of a comma followed by a space; a separator period is composed of a period followed by a space; a separator semicolon is composed of a semicolon followed by a space.

## Separators

The separator period must be used only to indicate the end of a sentence, or as shown in formats. The separator comma and separator semicolon can be used anywhere the separator space is used.

- In the **Identification Division**, each paragraph must end with a separator period.
- In the **Environment Division**, the SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, and I-O-CONTROL paragraphs must each end with a separator period. In the FILE-CONTROL paragraph, each File-Control entry must end with a separator period.
- In the **Data Division**, File (FD), Sort/Merge file (SD), and data description entries must each end with a separator period.
- In the **Procedure Division**, separator commas or separator semicolons can separate statements within a sentence, and operands within a statement. Each sentence and each procedure must end with a separator period.

**Parentheses { ( } ... { ) }**
Except in pseudo-text, parentheses can appear only in balanced pairs of left and right parentheses. They delimit subscripts, a list of function arguments, reference-modifiers, arithmetic expressions, or conditions.

**Colon { : }**
The colon is a separator and is required when shown in general formats.

**Quotation marks {"} . . . {"}**
An opening quotation mark must be immediately preceded by a space or a left parenthesis. A closing quotation mark must be immediately followed by a separator (space, comma, semicolon, period, right parenthesis, or pseudo-text delimiter). Quotation marks must appear as balanced pairs. They delimit nonnumeric literals, except when the literal is continued (see "Continuation Lines" on page 37).

x **Apostrophes {'} ... {'}**
x An opening apostrophe must be immediately preceded by a space or a left paren-
x thesis. A closing apostrophe must be immediately followed by a separator (space,
x comma, semicolon, period, or right parenthesis). Apostrophes must appear as bal-
x anced pairs. They delimit nonnumeric literals, except when the literal is continued
x (see "Continuation Lines" on page 37).

**Pseudo-text delimiters {b==} . . . {==b}**
An opening pseudo-text delimiter must be immediately preceded by a space. A closing pseudo-text delimiter must be immediately followed by a separator (space, comma, semicolon, or period). Pseudo-text delimiters must appear as balanced pairs. They delimit pseudo-text. (See "COPY Statement" on page 486.)

**Note:** Any punctuation character included in a PICTURE character-string, a comment character-string, or a nonnumeric literal is not considered as a punctuation character, but rather as part of the character-string or literal.

# Sections and Paragraphs

Sections and paragraphs define a program. They are subdivided into clauses and statements. For more information on sections, paragraphs, and statements, see “Procedures” on page 221.

## Statements and Clauses

Unless the associated rules explicitly state otherwise, each required clause or statement must be written in the sequence shown in its format. If optional clauses or statements are used, they must be written in the sequence shown in their formats. These rules are true even for clauses and statements treated as comments.

The grammatical hierarchy follows this form:

- Identification Division
  - Paragraphs
    - Entries
      - Clauses
- Environment Division
  - Sections
    - Paragraphs
      - Entries
        - Clauses
          - Phrases
- Data Division
  - Sections
    - Entries
      - Clauses
        - Phrases
- Procedure Division
  - Sections
    - Paragraphs
      - Sentences
        - Statements
          - Phrases

### Entries

An **entry** is a series of clauses ending with a separator period. Entries are constructed in the Identification, Environment, and Data Divisions.

### Clauses

A **clause** is an ordered set of consecutive COBOL character-strings that specifies an attribute of an entry. Clauses are constructed in the Identification, Environment, and Data Divisions.


© Copyright IBM Corp. 1991, 1996

## Sentences

A **sentence** is a sequence of one or more statements, ending with a separator period. Sentences are constructed in the Procedure Division.

## Statements

A **statement** is a valid combination of a COBOL verb and its operands. It specifies an action to be taken by the object program. Statements are constructed in the Procedure Division. For descriptions of the different types of statements, see:

- “Imperative Statements” on page 245
- “Conditional Statements” on page 246
- “Scope of Names” on page 41
- “Compiler-Directing Statement” on page 482

## Phrases

Each clause or statement in the program can be subdivided into smaller units called **phrases**.

# Reference Format

COBOL programs **must** be written in the COBOL reference format. Figure 1 shows the reference format for a COBOL source line.

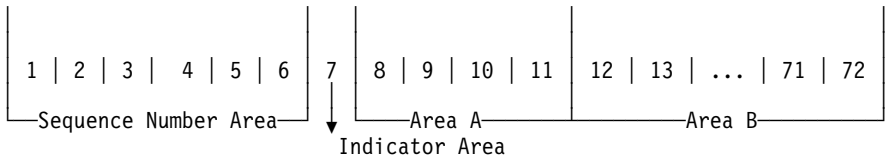



*Figure 1. Reference Format for COBOL Source Line*

The following areas are described below in terms of a 72-character line:

**Sequence Number Area**
Columns 1 through 6

**Indicator Area**
Column 7

**Area A**
Columns 8 through 11

**Area B**
Columns 12 through 72

## Sequence Number Area

The sequence number area may be used to label a source statement line. The content of this area may consist of any character in the character set of the computer.

## Indicator Area

Use the indicator area to specify:

- The continuation of words or nonnumeric literals from the previous line onto the current line
- The treatment of text as documentation
- Debugging lines

See “Continuation Lines” on page 37, “Comment Lines” on page 39, and “Debugging Lines” on page 40.

The indicator area can be used for source listing formatting. A slash (“/”) placed in the indicator column will cause the compiler to start a new page for the source listing, and the corresponding source record to be treated as a comment. The effect may be dependent on the LINECOUNT compiler option. See the “LINECOUNT” compiler option in the *IBM COBOL Programming Guide* for your platform.


© Copyright IBM Corp. 1991, 1996

## Area A

The following items must begin in Area A:

- Division header
- Section header
- Paragraph header or paragraph name
- Level indicator or level-number (01 and 77)
- DECLARATIVES and END DECLARATIVES
- End program, end class, and end method header

### Division Header

A division header is a combination of words, followed by a separator period, that indicates the beginning of a division:

```
IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.
```

A division header (except when a USING phrase is specified with a Procedure Division header) must be immediately followed by a separator period. Except for the USING phrase, no text may appear on the same line.

### Section Header

In the Environment and Procedure Divisions, a section header indicates the beginning of a series of paragraphs; for example:

```
INPUT-OUTPUT SECTION.
```

In the Data Division, a section header indicates the beginning of an entry; for example:

```
FILE SECTION.

LINKAGE SECTION.

WORKING-STORAGE SECTION.
```

A section header must be immediately followed by a separator period.

### Paragraph Header or Paragraph Name

A paragraph header or paragraph name indicates the beginning of a paragraph.

In the Environment Division, a paragraph consists of a paragraph header followed by one or more entries. For example:

```
OBJECT-COMPUTER.  computer-name
```

In the Procedure Division, a paragraph consists of a paragraph-name followed by one or more sentences.

## Area A

### Level Indicator (FD and SD) or Level-Number (01 and 77)

A level indicator can be either FD or SD. It must begin in Area A and be followed by a space. (See "File Section" on page 144.) A level-number that must begin in Area A is a 1- or 2-digit integer with a value of 01 or 77. It must be followed by a space or separator period.

### DECLARATIVES and END DECLARATIVES

DECLARATIVES and END DECLARATIVES are key words that begin and end the declaratives part of the source program.

In the Procedure Division, each of the key words DECLARATIVES and END DECLARATIVES must begin in Area A and be followed immediately by a separator period; no other text may appear on the same line. After the key words END DECLARATIVES, no text may appear before the following section header. (See "Declaratives" on page 220.)

### End Program, End Class, and End Method Headers

The "end" headers are a combination of words, followed by a separator period, that indicate the end of a COBOL source program, class definition, or method definition. For example:

```
END PROGRAM PROGRAM-NAME.
END CLASS CLASS-NAME.
END METHOD METHOD-NAME.
```

**For Programs**

Program-name must be identical to the program-name of the corresponding PROGRAM-ID paragraph. Every COBOL program, except an outermost program that contains no nested programs and is not followed by another batch program, must end with an END PROGRAM header.

**For Classes**

Class-name must be identical to the class-name of the corresponding CLASS-ID paragraph.

**For Methods**

Method-name must be identical to the method-name of the corresponding METHOD-ID paragraph.

## Area B

The following items must begin in Area B:

- Entries, sentences, statements, clauses
- Continuation lines

### Entries, Sentences, Statements, Clauses

The first entry, sentence, statement, or clause begins on either the same line as the header or paragraph-name it follows, or in Area B of the next nonblank line that is not a comment line. Successive sentences or entries either begin in Area B of the same line as the preceding sentence or entry or in Area B of the next nonblank line that is not a comment line.

Within an entry or sentence, successive lines in Area B may have the same format, or may be indented to clarify program logic. The output listing is indented only if the input statements are indented. Indentation does not affect the meaning of the program. The programmer can choose the amount of indentation, subject only to the restrictions on the width of Area B. See also “Sections and Paragraphs” on page 32.

### Continuation Lines

Any sentence, entry, clause, or phrase that requires more than one line can be continued in Area B of the next line that is neither a comment line nor a blank line. The line being continued is a **continued line**; the succeeding lines are **continuation lines**. Area A of a continuation line must be blank.

If there is no hyphen (-) in the indicator area (column 7) of a line, the last character of the preceding line is assumed to be followed by a space.

x DBCS literals and user-defined words containing multi-byte characters cannot be con-
x tinued.

x Both characters making up the opening delimiter must be on the same line for the:

x - Hexadecimal notation of a nonnumeric literal (X" or X')
x - Hexadecimal notation of a null-terminated nonnumeric literal (Z" or Z')

If there is a hyphen in the indicator area of a line, the first nonblank character of this continuation line immediately follows the last nonblank character of the continued line without an intervening space.

If the continued line contains a nonnumeric literal without a closing quotation mark, all spaces at the end of the continued line (through column 72) are considered to be part of the literal. The continuation line must contain a hyphen in the indicator area, and the first nonblank character must be a quotation mark. The continuation of the literal begins with the character immediately following the quotation mark.

If the last character on the continued line of a nonnumeric literal is a single quotation mark in column 72, the continuation line must start with two consecutive quotation

marks. This will result in a single quotation mark as part of the value of the nonnumeric literal.

x If the last character on the continued line of a nonnumeric literal is a single quotation
x mark in Area B, the continuation line may start with a single quotation mark. This will
x result in two consecutive nonnumeric literals instead of one continued nonnumeric
x literal.

Both characters making up the pseudo-text delimiter separator "==" must be on the same line.

To continue a literal such that the continued lines and the continuation lines are part of one literal:

- Code a hyphen in the indicator area of each continuation line.
- Do not terminate the continued lines with a single quotation mark followed by a space.
- Code the literal value using all columns of the continued lines, up to and including column 72.
- Code a quotation mark before the first character of the literal on each continuation line.
- Terminate the last continuation line with a single quotation mark followed by a space.

Given the following examples, the number and size of literals created are as follows:

- Literal 000001 is interpreted as one literal that is 120 bytes long. Each character between the starting quotation mark and up to and including column 72 of continued lines are counted as part of the literal.
- Literal 000005 is interpreted as one literal that is 140 bytes long. The blanks at the end of each continued line are counted as part of the literal because the continued lines do not end with a quotation mark.
- Literal 000010 is interpreted as three separate literals, each having a length of 50, 50, and 20, respectively. The quotation mark with the following space terminates the continued line. Only the characters within the quotation marks are counted as part of the literals. Literal 000010 is not valid as a VALUE clause literal for non-level 88 data items.

### Example

```
|...+.*..1....+....2....+....3....+....4....+....5....+....6....+....7..
000001               "AAAAAAAAAABBBBBBBBBCCCCCCCCCDDDDDDDDDEEEEEEEEEE
      -              "GGGGGGGGGGHHHHHHHHHHIIIIIIIIIJJJJJJJJJKKKKKKKKKK
      -    "LLLLLLLLLMMMMMMMMM"

000005     "AAAAAAAAAABBBBBBBBBCCCCCCCCCDDDDDDDDDEEEEEEEEEE
      -    "GGGGGGGGGGHHHHHHHHHIIIIIIIIIJJJJJJJJJKKKKKKKKKK
      -    "LLLLLLLLLMMMMMMMMM"

000010     "AAAAAAAAAABBBBBBBBBCCCCCCCCCDDDDDDDDDEEEEEEEEEE"
      -    "GGGGGGGGGGHHHHHHHHHIIIIIIIIIJJJJJJJJJKKKKKKKKKK"
      -    "LLLLLLLLLMMMMMMMMM"
```

**Note:** To code a continued literal where the length of each continued segment of the literal is less than the length of Area-B, adjust the starting column such that the last character of the continued segment is in column 72.

## Area A or Area B

The following items may begin in either Area A or Area B:

- Level-numbers
- Comment lines
- Compiler-directing statements
- Debugging lines
- Pseudo-text

## Level-Numbers

A level-number that may begin in Area A or B is a 1- or 2-digit integer with a value of 02 through 49; 66, or 88. A level-number that must begin in Area A is a 1- or 2-digit integer with a value of 01 or 77. It must be followed by a space or a separator period. For more information, see “Level-Numbers” on page 158.

## Comment Lines

A **comment line** is any line with an asterisk (*) or slash (/) in the indicator area (column 7) of the line. The comment may be written anywhere in Area A and Area B of that line, and may consist of any combination of characters from the character set of the computer. A comment line may be placed anywhere in the program following the Identification Division header.

x Comment lines are permitted to appear before the Identification Division, but they must
x follow any control cards (for example, PROCESS or CBL).

x **Note:** Comments intermixed with control cards could nullify some of the control cards
x and cause them to be diagnosed as errors.

Multiple comment lines are allowed. Each must begin with either an asterisk (*) or a slash (/) in the indicator area.

### Area A or Area B

An asterisk (*) comment line is printed on the next available line in the output listing. The effect may be dependent on the LINECOUNT compiler option. See “LINECOUNT” compiler option in the *IBM COBOL Programming Guide* for your platform. A slash (/) comment line is printed on the first line of the next page, and the current page of the output listing is ejected.

The compiler treats a comment line as documentation, and does not check it syntactically.

### Compiler-Directing Statements:

Most compiler-directing statements may start in either Area A or Area B, including COPY and REPLACE.

x As an IBM extension BASIS, CBL (PROCESS), *CBL (*CONTROL), DELETE, EJECT,
x INSERT, SKIP/2/3, and TITLE can also start in Area A or Area B.

### x Compiler Directives

x Compiler directives can start only in Area B. Currently, the only compiler directive is
x CALLINTERFACE.

### Debugging Lines

A **debugging line** is any line with a `'D'` (or `'d'`) in the indicator area of the line. Debugging lines can be written in the Environment Division (after the OBJECT-COMPUTER paragraph), the Data Division, and the Procedure Division. If a debugging line contains only spaces in Area A and Area B, it is considered a blank line.

See “WITH DEBUGGING MODE” on page 85.

### Pseudo-Text

The character-strings and separators comprising **pseudo-text** may start in either Area A or Area B. If, however, there is a hyphen in the indicator area (column 7) of a line which follows the opening pseudo-text delimiter, Area A of the line must be blank, and the rules for continuation lines apply to the formation of text words.

### Blank Lines

A **blank line** contains nothing but spaces from column 7 through column 72. A blank line may appear anywhere in a program.

# Scope of Names

A COBOL resource is any resource in a COBOL program that is referenced via a user-defined word. You can use names to identify COBOL resources. This section describes COBOL names and their scope. It explains the range of where the names can be referenced and the range of their usability and accessibility.

## Types of Names

In addition to identifying a resource, a name can have global or local attributes. Some names are always global, some names are always local, and some names are either local or global depending on specifications in the program in which the names are declared.

**For Programs**

A **global name** can be used to refer to the resource with which it is associated both:

- From within the program in which the global name is declared
- From within any other program that is contained in the program that declares the global name

You use the GLOBAL clause in the data description entry to indicate that a name is global. For more information on using the GLOBAL clause, see "GLOBAL Clause" on page 161.

A **local name** can be used only to refer to the resource with which it is associated from within the program in which the local name is declared.

By default, if a data-name, a file-name, a record-name, or a condition-name declaration in a data description entry does not include the GLOBAL clause, the name is local.

x **For Classes and Methods**

x Names declared in a class definition are global to all the methods contained in that
x class definition. All names declared in methods are implicitly local.

**Note:** Specific rules sometimes prohibit specifying the GLOBAL clause for certain data description, file description, or record description entries.

The following list indicates the names you can use and whether the name can be local or global:

**data-name**

Data-name assigns a name to a data item.

A data-name is global if the GLOBAL clause is specified either in the data description entry that declares the data-name, or in another entry to which that data description entry is subordinate.

© Copyright IBM Corp. 1991, 1996

## Scope of Names

**file-name**
File-name assigns a name to a file connector.

A file-name is global if the GLOBAL clause is specified in the file description entry for that file-name.

**record-name**
Record-name assigns a name to a record.

A record-name is global if the GLOBAL clause is specified in the record description that declares the record-name, or in the case of record description entries in the File Section, if the GLOBAL clause is specified in the file description entry for the file name associated with the record description entry.

**condition-name**
Condition-name associates a value with a conditional variable.

A condition-name that is declared in a data description entry is global if that entry is subordinate to another entry that specifies the GLOBAL clause.

A condition-name that is declared within the Configuration Section is always global.

**program-name**
Program-name assigns a name to a program, either external or internal (nested). For more information, see “Conventions for Program-Names” on page 66.

A program-name is neither local nor global. For more information, see “Conventions for Program-Names” on page 66.

x **method-name**
x Method-name assigns a name to a method. A method-name is neither local nor
x global.

**section-name**
Section-name assigns a name to a section in the Procedure Division.

A section-name is always local.

**paragraph-name**
Paragraph-name assigns a name to a paragraph in the Procedure Division.

A paragraph-name is always local.

**basis-name**
Basis-names are treated consistently as defined for text-names without the library-name qualification.

**library-name**
MVS/VM Under MVS and VM, library-name specifies the COBOL library that the compiler uses for a given source program compilation.

A library-name is external to the program and can be referenced by any COBOL program if the compiler system supports the associated library and the entities referenced are known to that system.

WRKSTN Under AIX, OS/2, and Windows, a library-name is used to identify the path for the library text.

If you specify library-name with a literal, it is treated as the actual path name. If you specify library-name with a user-defined word, the name is used as an environment variable and the value of the environment variable is used for the path names(s) to locate the COPY text. For details on path names, see "COPY Statement" on page 486.

**text-name**

MVS/VM Under MVS and VM, text-name assigns a name to library text. A text-name is external to the program and can be referenced by any COBOL program if the compiler system supports the associated library and the entities referenced are known to that system.

WRKSTN Under AIX, OS/2, and Windows, a text-name is used to identify the file for the COPY text. For details, see "COPY Statement" on page 486.

**alphabet-name**
Alphabet-name assigns a name to a specific character set and/or collating sequence in the SPECIAL-NAMES paragraph of the Environment Division.

An alphabet-name is always global.

**class-name**
Class-name assigns a name to the proposition in the SPECIAL-NAMES paragraph of the Environment Division for which a truth value can be defined.

A class-name is always global.

**object-oriented class-name**
Object-oriented class-name assigns a name to a class, subclass, or metaclass. An object-oriented class-name is always global.

**object-oriented class Working-Storage**
Object-oriented class Working-Storage data items are always global to the methods contained in the class definition. They are accessible from any contained method.

**mnemonic-name**
Mnemonic-name assigns a user-defined word to an implementer-name.

A mnemonic-name is always global.

**symbolic-character**
Symbolic-character specifies a user-defined figurative constant.

A symbolic-name is always global.

**index-name**
Index-name assigns a name to an index associated with a specific table.

If a data item possessing the GLOBAL attribute includes a table accessed with an index, that index also possesses the GLOBAL attribute. In addition, the scope of that index-name is identical to the scope of the data-name that includes the table.

## External and Internal Resources

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 can be **external** or **internal** to the program or method in which the resource is declared.

A data item or file connector is external if the storage associated with that resource is associated with the run unit rather than with any particular program or method within the run unit. An external resource can be referenced by any program or method in the run unit that describes the resource. References to an external resource from different programs or methods using separate descriptions of the resource are always to the same resource. In a run unit, there is only one representation of an external resource.

A resource is internal if the storage associated with that resource is associated only with the program or method that describes the resource.

External and internal resources can have either global or local names.

A data 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 program or method in which it is described.

Two programs or methods in a run unit can reference the same file connector in the following circumstances:

- An external file connector can be referenced from any program or method that describes that file connector.
- If a program is contained within another program, both programs can refer to a global 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.

Two programs or methods in a run unit can reference common data in the following circumstances:

- The data content of an external data record can be referenced from any program or method provided that program or method has described that data record.
- If a program is contained within another program, both programs can refer to data possessing the global attribute either in the program or in any program that directly or indirectly contains the containing program.

The data records described as 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 to the program or method describing the file-name. If the EXTERNAL clause is included in the file description entry, the data records and the data items attain the external attribute.

## Resolution of Names

When a program, program B, is directly contained within another program, program A, both programs can define a condition-name, a data-name, a file-name, or a record-name using the same user-defined word. When such a duplicated name is referenced in program B, the following steps determine the referenced resource (note, these rules also apply to classes and contained methods):

1. The referenced resource is identified from the set of all names which are defined in program B and all global names defined in program A and in any programs which directly or indirectly contain program 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 resource is identified.
2. If only one resource is identified, it is the referenced resource.
3. If more than one resource is identified, no more than one of them can have a name local to program B. If zero or one of the resources has a name local to program B, the following applies:
   - If the name is declared in program B, the resource in program B is the referenced resource.
   - If the name is not declared in program B, the referenced resource is:
     – The resource in program A if the name is declared in program A.
     – The resource in the containing program if the name is declared in the program containing program A.

     This rule is applied to further containing programs until a valid resource is found.

# Referencing Data Names, Copy Libraries, and Procedure Division Names

References can be made to external and internal resources. References to data and procedures can be either explicit or implicit. This section contains the rules for qualification and for explicit and implicit data references.

## Uniqueness of Reference

Every user-defined name in a COBOL program is assigned by the user to name a resource for solving a data processing problem. To use a resource, a statement in a COBOL program must contain a reference which uniquely identifies that resource. To ensure uniqueness of reference, a user-defined name can be qualified, subscripted, or reference-modified.

When the same name has been assigned in separate programs to two or more occurrences of a resource of a given type, and when qualification by itself does not allow the references in one of those programs to differentiate between the identically named resources, then certain conventions that limit the scope of names apply. The conventions ensure that the resource identified is that described in the program containing the reference. For more information on resolving program-names, see “Resolution of Names” on page 45.

Unless otherwise specified by the rules for a statement, any subscripts and reference modification are evaluated only once as the first step in executing that statement.

### Qualification

A name can be made unique if it exists within a hierarchy of names by specifying one or more higher-level names in the hierarchy. The higher-level names are called **qualifiers**, and the process by which such names are made unique is called **qualification**.

Qualification is specified by placing one or more phrases after a user-specified name, with each phrase made up of the word IN or OF followed by a qualifier (IN and OF are logically equivalent).

In any hierarchy, the data name associated with the highest level must be unique if it is referenced, and cannot be qualified.

You must specify enough qualification to make the name unique; however, it is not always necessary to specify all the levels of the hierarchy. For example, if there is more than one file whose records contain the field `EMPLOYEE-NO`, but only one of the files has a record named `MASTER-RECORD`:

- `EMPLOYEE-NO OF MASTER-RECORD` sufficiently qualifies `EMPLOYEE-NO`
- `EMPLOYEE-NO OF MASTER-RECORD OF MASTER-FILE` is valid but unnecessary.


© Copyright IBM Corp. 1991, 1996

### Qualification Rules

The rules for qualifying a name are:

- A name can be qualified even though it does not need qualification except in a REDEFINES clause, in which case it **must not** be qualified.
- Each qualifier must be of a higher level than the name it qualifies, and must be within the same hierarchy.
- If there is more than one combination of qualifiers that ensures uniqueness, then any of these combinations can be used.

## Data Attribute Specification

**Explicit data attributes** are those you specify in actual COBOL coding.

**Implicit data attributes** are default values. If you do not explicitly code a data attribute, the compiler assumes a default value.

For example, you need not specify the USAGE of a data item. If it is omitted and the symbol N is not specified in the PICTURE clause, the default is USAGE DISPLAY, which is the implicit data attribute.

x When PICTURE clause symbol N is used, USAGE DISPLAY-1 is assumed (for DBCS
x items). If, however, you specify USAGE DISPLAY in COBOL coding, it becomes an explicit data attribute.

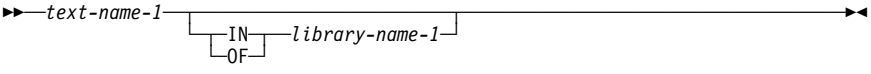
## Identical Names

When programs are directly or indirectly contained within other programs, each program can use identical user-defined words to name resources. With identically-named resources, a program will reference the resource which that program describes rather than the same-named resource described in another program, even when it is a different type of user-defined word.

x These same rules apply to classes and their contained methods.

## References to COPY Libraries

**Format**



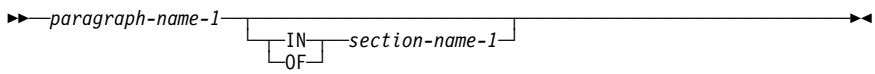

x If more than one COBOL library is available to the compiler during compilation, as an
x IBM extension, text-name-1 need not be qualified each time it is referenced; a qualifica-
x tion of SYSLIB is assumed.

For rules on referencing COPY libraries, see “COPY Statement” on page 486.

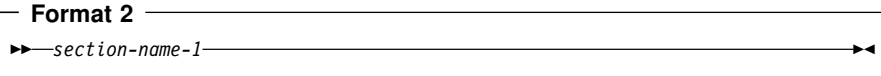
## Uniqueness of Reference

## References to Procedure Division Names

**Format 1**




**Format 2**




Procedure Division names that are explicitly referenced in a program must be unique within a section. A section-name, described under “Procedures” on page 221, is the highest and only qualifier available for a paragraph-name and must be unique if referenced.

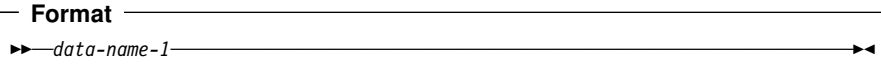
If explicitly referenced, a paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to within the section in which it appears. A paragraph-name or section-name appearing in a program cannot be referenced from any other program.

## References to Data Division Names

### Simple Data Reference

The most basic method of referencing data items in a COBOL program is **simple data reference**, which is data-name-1 without qualification, subscripting, or reference modification. Simple data reference is used to reference a single elementary or group item.

**Format**




**data-name-1**

Can be any data description entry.

Data-name-1 must be unique in a program.

### Identifier

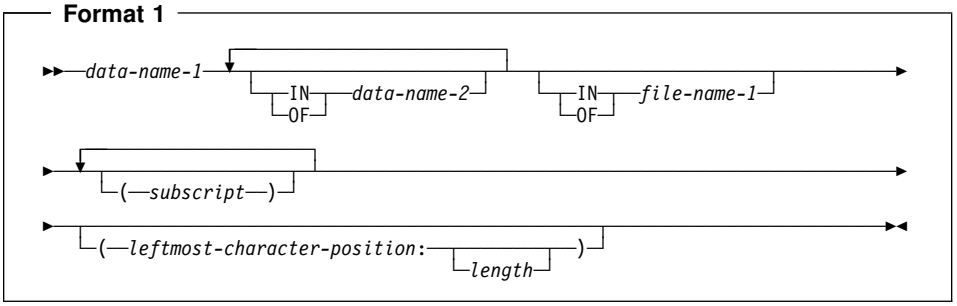
When used in a syntax diagram in this manual, the term **identifier** refers to a valid combination of a data-name or function-identifier with its qualifiers, subscripts, and reference-modifiers as required for uniqueness of reference. Rules for identifiers associated with a format can, however, specifically prohibit qualification, subscripting, or reference-modification.

The term **data-name** refers to a name that must not be qualified, subscripted, or reference modified, unless specifically permitted by the rules for the format.

- For a description of qualification, see “Qualification” on page 46.

- For a description of subscripting, see “Subscripting” on page 51.
- For a description of reference modification, see “Reference Modification” on page 54.

**Format 1**

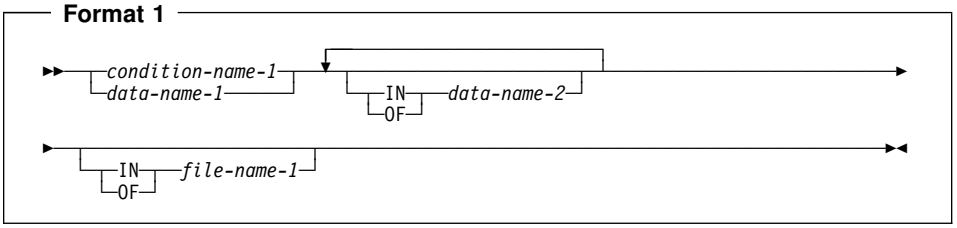



**data-name-1, data-name-2**
Can be a record-name.

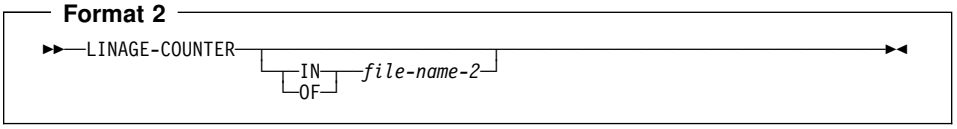
**file-name-1**
Must be identified by an FD or SD entry in the Data Division.

File-name-1 must be unique within this program.

**Format 1**




**Format 2**




**data-name-1, data-name-2**
Can be a record-name.

**condition-name-1**
Can be referenced by statements and entries either in that program containing the Configuration Section or in a program contained within that program.

**file-name-1**
Must be identified by an FD or SD entry in the Data Division.

Must be unique within this program.

## Uniqueness of Reference

**LINAGE-COUNTER**
Must be qualified each time it is referenced if more than one file description entry containing a LINAGE clause has been specified in the source program.

**file-name-2**
Must be identified by the FD or SD entry in the Data Division. File-name-2 must be unique within this program.

Duplication of data-names must not occur in those places where the data-name cannot be made unique by qualification.

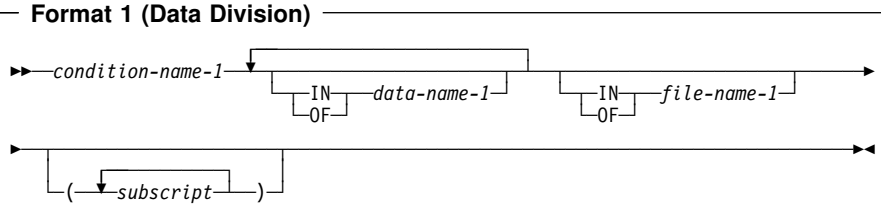
In the same program, the data-name specified as the subject of the entry whose level-number is 01 that includes the EXTERNAL clause must not be the same data-name specified for any other data description entry that includes the EXTERNAL clause.

In the same Data Division, the data description entries for any two data items for which the same data-name is specified must not include the GLOBAL clause.

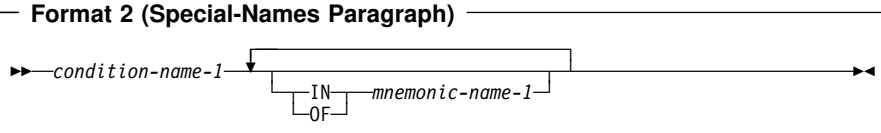
Data Division names that are explicitly referenced must either be uniquely defined or made unique through qualification. Unreferenced data items need not be uniquely defined. The highest level in a data hierarchy must be uniquely named, if referenced. This is a data item associated with a level indicator (FD or SD in the File Section) or with a level-number 01. Data items associated with level-numbers 02 through 49 are successively lower levels of the hierarchy.

## Condition-name

**Format 1 (Data Division)**




**Format 2 (Special-Names Paragraph)**




**condition-name-1**
Can be referenced by statements and entries either in the program containing the definition of condition-name-1, or in a program contained within that program.

If explicitly referenced, a condition-name must be unique or be made unique through qualification and/or subscripting except when the scope of names conventions by themselves ensure uniqueness of reference.

If qualification is used to make a condition-name unique, the associated conditional variable may be used as the first qualifier. If qualification is used, the hierarchy of names associated with the conditional variable itself must be used to make the condition-name unique.

If references to a conditional variable require subscripting, reference to any of its condition-names also requires the same combination of subscripting.

In the general format of the chapters that follow, "condition-name" refers to a condition-name qualified or subscripted, as necessary.

**data-name-1**
Can be a record-name.

**file-name-1**
Must be identified by an FD or SD entry in the Data Division.

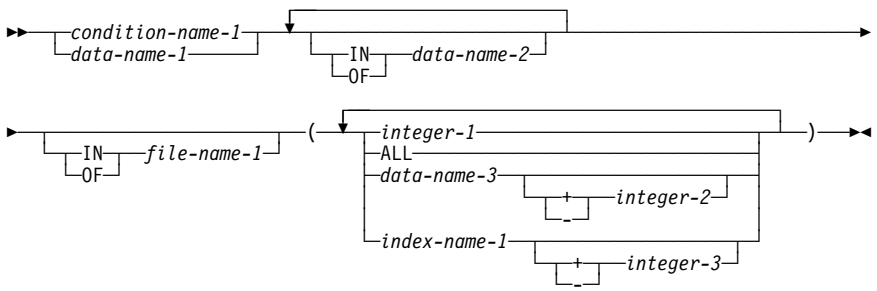
File-name-1 must be unique within this program.

**mnemonic-name-1**
For information on acceptable values for mnemonic-name-1, see "SPECIAL-NAMES Paragraph" on page 88.

## Subscripting

**Subscripting** is a method of providing table references through the use of subscripts. A **subscript** is a positive integer whose value specifies the occurrence number of a table element.

**Format**




**condition-name-1**
The conditional variable for condition-name-1 must contain an OCCURS clause or must be subordinate to a data description entry which contains an OCCURS clause.

**data-name-1**
Must contain an OCCURS clause or must be subordinate to a data description entry which contains an OCCURS clause.

## Uniqueness of Reference

**data-name-2, file-name-1**
Must name data items or records that contain data-name-1.

**integer-1**
Can be signed. If signed, it must be positive.

**data-name-3**
Must be a numeric elementary item representing an integer.

Data-name-3 can be qualified.

**index-name-1**
Corresponds to a data description entry in the hierarchy of the table being referenced which contains an INDEXED BY phrase specifying that name.

**integer-2, integer-3**
Cannot be signed.

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 must equal the number of dimensions in the table whose element is being referenced. That is, there must 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.

For example, if `TABLE-THREE` is defined as:

```
01  TABLE-THREE.
    05  ELEMENT-ONE OCCURS 3 TIMES.
        10  ELEMENT-TWO OCCURS 3 TIMES.
            15  ELEMENT-THREE OCCURS 2 TIMES    PIC X(8).
```

a valid subscripted reference to `TABLE-THREE` is:

```
ELEMENT-THREE (2 2 1)
```

Subscripted references may also be reference modified. See the third example on page 56.

A reference to an item must not be subscripted unless the item is a table element **or** an item or condition-name associated with a table element.

Each table element reference must be subscripted except when such reference appears:

- In a USE FOR DEBUGGING statement
- As the subject of a SEARCH statement
- In a REDEFINES clause
- In the KEY is phrase of an OCCURS clause

The lowest permissible occurrence number represented by a subscript is 1. The highest permissible occurrence number in any particular case is the maximum number of occurrences of the item as specified in the OCCURS clause.

## Subscripting Using Data-Names

When a data-name is used to represent a subscript, it can be used to reference items within different tables. These tables need not have elements of the same size. The same data-name can appear as the only subscript with one item and as one of two or more subscripts with another item. A data-name subscript can be qualified; it cannot be subscripted or indexed. For example, valid subscripted references to `TABLE-THREE` — assuming that `SUB1`, `SUB2`, and `SUB3` are all items subordinate to `SUBSCRIPT-ITEM` — include:

```
ELEMENT-THREE (SUB1 SUB2 SUB3)


ELEMENT-THREE IN TABLE-THREE (SUB1 OF SUBSCRIPT-ITEM,
    SUB2 OF SUBSCRIPT-ITEM, SUB3 OF SUBSCRIPT-ITEM)
```

## Subscripting Using Index-Names (Indexing)

Indexing allows such operations as table searching and manipulating specific items. To use indexing you associate one or more index-names with 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. At run time, the contents of the index corresponds to an occurrence number for that specific dimension of the table with which the index is associated.

The initial value of an index at run time is undefined, and the index must be initialized before it is used as a subscript. An initial value is assigned to an index with one of the following:

- The PERFORM statement with the VARYING phrase
- The SEARCH statement with the ALL phrase
- The SET statement

## Uniqueness of Reference

The use of an integer 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.

| As an IBM extension, an index-name can be used to reference any table. However,
| the table element length of the table being referenced and of the table that the index-
| name is associated with should match. Otherwise, the reference will not be to the
| same table element in each table, and you might get run-time errors.

Data that is arranged in the form of a table is often searched. The SEARCH statement provides facilities for producing serial and non-serial searches. It is used to search for a table element that satisfies a specific condition and to adjust the value of the associated index to indicate that table element.

To be valid during execution, an index value must correspond to a table element occurrence of neither less than one, nor greater than the highest permissible occurrence number.

For more information on index-names, see “INDEXED BY Phrase” on page 166.

### Relative Subscripting

In **relative subscripting**, the name of a table element is followed by a subscript of the form data-name or index-name followed by the operator + or -, and an unsigned integer literal.

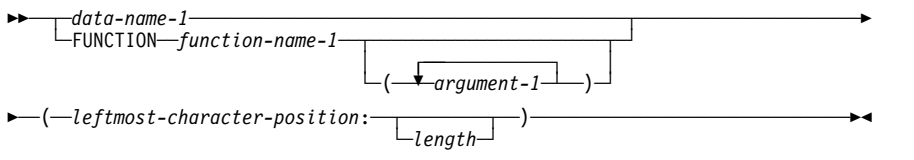
x As an IBM extension, the integer can be positively signed.

The operators + and - must be preceded and followed by a space. The value of the subscript used is the same as if the index-name or data-name had been set up or down by the value of the integer. The use of relative indexing does not cause the program to alter the value of the index.

## Reference Modification

Reference modification defines a data item by specifying a leftmost character and optional length for the data item.

**Format**




**data-name-1**

Must reference a data item whose usage is DISPLAY.

Data-name-1 can be qualified or subscripted.

**leftmost-character-position**
Must be an arithmetic expression. The evaluation of *leftmost-character-position* must result in a positive nonzero integer that is less than or equal to the number of characters in the data item referenced by data-name-1.

**length**
Must be an arithmetic expression.

The sum of *leftmost-character-position* and *length* minus the value one must be less than or equal to the number of characters in data-name-1. If *length* is omitted, than the length used will be equal to the number of characters in data-name-1 plus one minus *leftmost-character-position*. When data-name-1 is a DISPLAY-1 data item, reference modification refers to the starting position and length of the data item being referenced in characters, not bytes. The evaluation of length must result in a positive nonzero integer.

Unless otherwise specified, reference modification is allowed anywhere an identifier referencing an alphanumeric data item is permitted.

Each character of a data item referenced by data-name-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 data-name-1 contains a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number within that data item.

If the data item referenced by data-name-1 is described as numeric, numeric-edited, alphabetic, or alphanumeric-edited, it is operated upon for purposes of reference modification as if it were redefined as an alphanumeric data item of the same size as the data item referenced by data-name-1.

Reference modification creates a unique data item which is a subset of the data item referenced by data-name-1 or by function-name-1 and its arguments, if any. This unique data item is considered an elementary data item without the JUSTIFIED clause.

When a function is reference-modified, the unique data item has the class and category of alphanumeric. When data-name-1 is reference-modified, the unique data item has the same class and category as that defined for the data item referenced by data-name-1; however, if the category of data-name-1 is numeric, numeric-edited, or alphanumeric-edited, the unique data item has the class and category alphanumeric.

If the category of data-name-1 is external floating-point, the unique data item has the class and category alphanumeric.

If length is not specified, the unique data item created extends from and includes the character identified by leftmost-character-position up to and including the rightmost character of the data item referenced by data-name-1.

## Uniqueness of Reference

### Evaluation of Operands

Reference modification for an operand is evaluated as follows:

- If subscripting is specified for the operand, the reference modification is evaluated immediately after evaluation of the subscript.
- If subscripting is not specified for the operand, the reference modification is evaluated at the time subscripting would be evaluated if subscripts had been specified.

### Reference Modification Examples

The following statement transfers the first 10 characters of the data-item referenced by WHOLE-NAME to the data-item referenced by FIRST-NAME.

```
77  WHOLE-NAME  PIC X(25).
77  FIRST-NAME  PIC X(10).
         .......

    MOVE WHOLE-NAME(1:10) TO FIRST-NAME.
```

The following statement transfers the last 15 characters of the data-item referenced by WHOLE-NAME to the data-item referenced by LAST-NAME.

```
77  WHOLE-NAME  PIC X(25).
77  LAST-NAME   PIC X(15).
         .......

    MOVE WHOLE-NAME(11:) TO LAST-NAME.
```

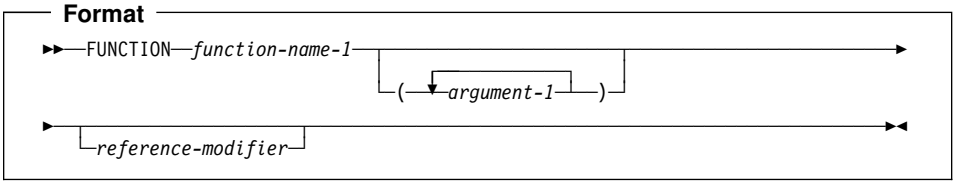
The following statement transfers the fourth and fifth characters of the third occurrence of TAB to the variable SUFFIX.

```
01  TABLE-1.
    02  TAB  OCCURS 10 TIMES  PICTURE X(5).
77  SUFFIX                   PICTURE X(2).
         .......

    MOVE TAB OF TABLE-1 (3) (4:2) TO SUFFIX.
```

## Function-Identifier

A function-identifier is a syntactically correct sequence of character strings and separators that uniquely references the data item resulting from the evaluation of a function.

**Format**




**argument-1**
Must be an identifier, literal (other than a figurative constant), or arithmetic expression.

For more information, see “Intrinsic Functions” on page 426.

**function-name-1**
Function-name-1 must be one of the Intrinsic Function names.

**reference-modifier**
May be specified only for functions of the category alphanumeric

A function-identifier that makes reference to an alphanumeric function may be specified anywhere that an identifier is permitted and where references to functions are not specifically prohibited, except as follows:

- As a receiving operand of any statement
- Where a data item is required to have particular characteristics (such as class and category, size, sign, and permissible values) and the evaluation of the function according to its definition and the particular arguments specified would not have these characteristics.

A function-identifier that makes reference to an integer or numeric function may be used wherever an arithmetic expression is allowed.

# Transfer of Control

In the Procedure Division, unless there is an **explicit** control transfer or there is no next executable statement, program flow transfers control from statement to statement in the order in which the statements are written. (See Note below.) This normal program flow is an **implicit** transfer of control.

In addition to the implicit transfers 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. The following examples show **implicit** transfers of control, overriding statement-to-statement transfer of control:

- After execution of the last statement of a procedure being executed under control of another COBOL statement, control implicitly transfers. (COBOL statements that control procedure execution are, for example: MERGE, PERFORM, SORT, and USE.) Further, if a paragraph is being executed under the control of a PERFORM statement which 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.
- During SORT or MERGE statement execution, control is implicitly transferred to an input or output procedure.
- During execution of any COBOL statement that causes execution of a declarative procedure, control is implicitly transferred to that procedure.
- At the end of execution of any declarative procedure, control is implicitly transferred back to the control mechanism associated with the statement that caused its execution.

COBOL also provides **explicit** control transfers through the execution of any procedure branching, program call, or conditional statement. (Lists of procedure branching and conditional statements are contained in “Statement Categories” on page 245.)

**Note:** The term “next executable statement” refers to the next COBOL statement to which control is transferred, according to the rules given above. There is no **next executable statement** under these circumstances:

- When the program contains no Procedure Division
- Following the last statement in a declarative section when the paragraph in which it appears is not being executed under the control of some other COBOL statement
- Following the last statement in a program or method when the paragraph in which it appears is not being executed under the control of some other COBOL statement in that program
- Following the last statement in a declarative section when the statement is in the range of an active PERFORM statement executed in a different section and this


© Copyright IBM Corp. 1991, 1996

last statement of the declarative section is not also the last statement of the procedure that is the exit of the active PERFORM statement

- Following a STOP RUN statement or EXIT PROGRAM statement that transfers control outside the COBOL program
- Following a GOBACK statement that transfers control outside the COBOL program
- Following an EXIT METHOD statement that transfers control outside the COBOL method.
- The end program or end method header

When there is no next executable statement and control is not transferred outside the COBOL program, the program flow of control is undefined unless the program execution is in the nondeclarative procedures portion of a program under control of a CALL statement, in which case an implicit EXIT PROGRAM statement is executed.

Similarly, if control reaches the end of the Procedure Division of a method, and there is no next executable statement, an implicit EXIT METHOD statement is executed.

## Transfer of Control

# Part 2. COBOL Source Unit Structure

**COBOL Program Structure** . . . . . 62
Nested Programs . . . . . 65

x **COBOL Class Definition Structure** . . . . . 68

x **COBOL Method Definition Structure** . . . . . 70

© Copyright IBM Corp. 1991, 1996

# COBOL Program Structure

A COBOL source program is a syntactically correct set of COBOL statements.

**Nested Programs**
A nested program is a program that is contained in another program. These contained programs can reference some of the resources of the programs that contain them. If program B is contained in program A, it is **directly** contained if there is no program contained in program A that also contains program B. Program B is **indirectly** contained in program A if there exists a program contained in program A that also contains program contained and containing programs, see B. For more information on nested programs “Nested Programs” on page 65, and the *Programming Guide* for your platform.

**Object Program**
An object program is a set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. An object program is generally the machine language result of the operation of a COBOL compiler on a source program.

**Run Unit**
A Run unit is one or more object programs that interact with one another and that function at object time as an entity to provide problem solutions.

**Sibling program**
Sibling programs are programs that are directly contained by the same program.

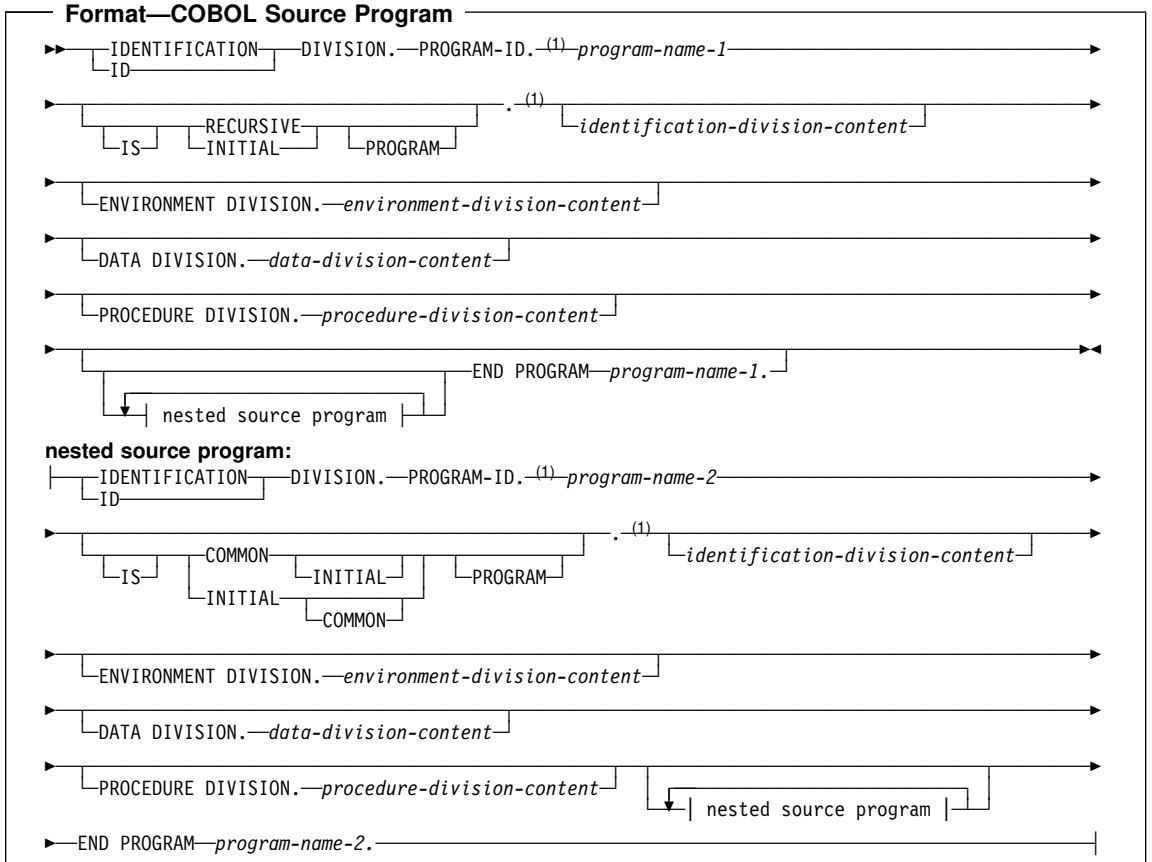
With the exception of the COPY and REPLACE statements and the end program header, the statements, entries, paragraphs, and sections of a COBOL source program are grouped into the following four divisions:

- Identification Division
- Environment Division
- Data Division
- Procedure Division

The end of a COBOL source program is indicated by the END PROGRAM header. If there are no nested programs, the absence of additional source program lines also indicates the end of a COBOL program.

Following is the format for the entries and statements that constitute a separately-compiled COBOL source program.

**Format—COBOL Source Program**



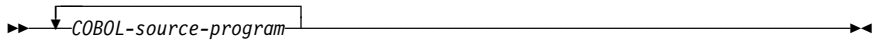

**Note:**

1 This separator period is optional as an IBM extension.

## COBOL Program Structure

A sequence of separate COBOL programs can also be input to the compiler. Following is the format for the entries and statements that constitute a sequence of source programs (batch compile).

**Format—Sequence of COBOL Source Programs**




**END PROGRAM program-name**

An end program header separates each program in the sequence of programs. The program-name must conform to the rules for forming a user-defined word. It must be identical to a program-name declared in a preceding PROGRAM-ID paragraph.

x Program-name can be a nonnumeric literal, but cannot be a figurative constant.
x The content of the literal must follow the rules for formation of program names.
x Any lowercase letters in this literal will be folded to uppercase.

An end program header is optional for the last program in the sequence only if that program does not contain any nested-source-programs.

## Nested Programs

A COBOL program can contain other COBOL programs, which in turn can contain still other COBOL programs. These contained programs are called nested programs. Nested programs can be **directly** or **indirectly** contained in the containing program.

A COBOL program may contain other COBOL programs. The contained (or nested) programs may themselves contain yet other programs. A contained program may be **directly** or **indirectly** contained within another program. Figure 2 describes a nested program structure with directly and indirectly contained programs.

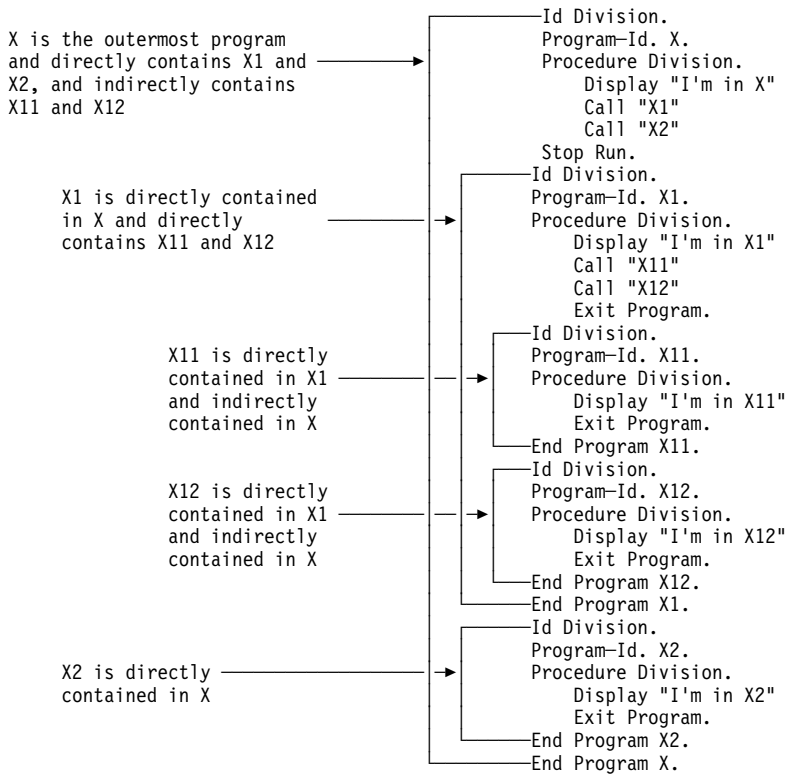



*Figure 2. Nested program structure with directly and indirectly contained programs*

# COBOL Program Structure

## Conventions for Program-Names

The program-name of a program is specified in the PROGRAM-ID paragraph of the program's Identification Division. A program-name can be referenced only by the CALL statement, the CANCEL statement, the SET statement, or the END PROGRAM header. Names of programs constituting a run unit are not necessarily unique, but when two programs in a run unit are identically named, at least one of the programs must be directly or indirectly contained within another separately compiled program that does not contain the other of those two programs.

A separately compiled program and all of its directly and indirectly contained programs must have unique program-names within that separately compiled program.

### Rules for Program-Names

The following rules regulate the scope of a program-name:

- If the program-name is that of a program which does not possess the COMMON attribute, and which is directly contained within another program, that program-name can be referenced only by statements included in that containing program.
- If the program-name is that of a program which does possess the COMMON attribute, and which is directly contained within another program, that program-name can be referenced only by statements included in that containing program and any programs directly or indirectly contained within that containing program, except that program possessing the COMMON attribute and any programs contained within it.
- If the program-name is that of a program which is separately compiled, that program-name can be referenced by statements included in any other program in the run unit, except programs it directly or indirectly contains.

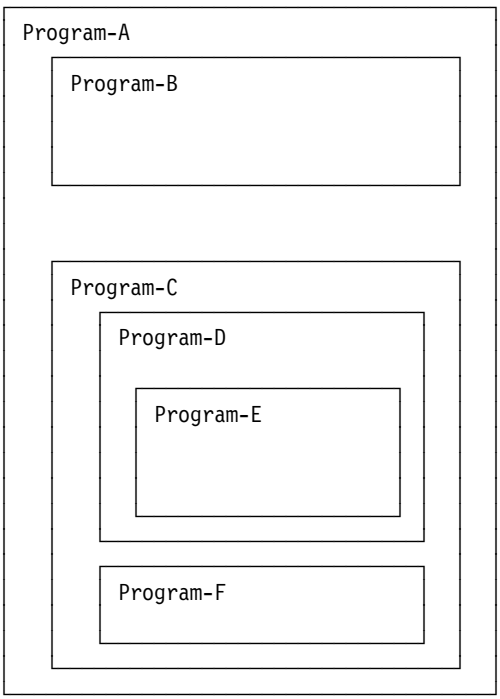
  The mechanism used to determine which program to call is as follows:

  – If one of two programs having the same name as that specified in the CALL statement is directly contained within the program that includes the CALL statement, that program is called.

  – If one of two programs having the same name as that specified in the CALL statement possesses the COMMON attribute and is directly contained within another program that directly or indirectly contains the program that includes the CALL statement, that common program is called unless the calling program is contained within that common program.

  – Otherwise, the separately compiled program is called.

The following rules apply to referencing a program-name of a program that is contained within another program. For this discussion, we will say that Program-A contains Program-B and Program-C, Program-C contains Program-D and Program-F, and Program-D contains Program-E.




If Program-D does not possess the COMMON attribute, then Program-D can only be referenced by the program that directly contains Program-D, that is, Program-C.

If Program-D does possess the COMMON attribute, then Program-D can be referenced by Program-C since it contains Program-D and by any programs contained in Program-C except for programs contained in Program-D. In other words, if Program-D possesses the COMMON attribute, Program-D can be referenced in Program-C and Program-F but not by statements in Program-E, Program-A or Program-B.

# COBOL Class Definition Structure

A COBOL class definition describes a class or a metaclass. A class definition constitutes a compilation unit.

**Class**
The entity that defines common behavior and implementation for zero, one, or more objects. The objects that share the same implementation are considered to be objects of the same class.

**Method**
Procedural code that defines one of the operations supported by an object, and that is executed by an INVOKE statement on that object.

**Instance Data**
Data defining the state of an object. The instance data introduced by a class is defined in the Working-Storage Section of the Data Division of the class definition. The state of an object also includes the state of the instance variables introduced by bases classes that are inherited by the current class. A separate copy of the instance data is created for each object instance.

**Subclass**
A class that inherits methods and instance data from another class. When two classes in an inheritance relationship are considered together, the subclass is the inheritor or inheriting class; the *super-class* is the inheritee or inherited class.

**Metaclass**
A special type of class whose instances are called class-objects. Class-objects are the run-time objects that represent SOM classes. Any class descended from SOMClass is a metaclass.

With the exception of the COPY and REPLACE statements and the END CLASS header, the statements, entries, paragraphs, and sections of a COBOL class definition are grouped into the following four divisions:

- Identification Division
- Environment Division (Configuration Section only)
- Data Division
- Procedure Division

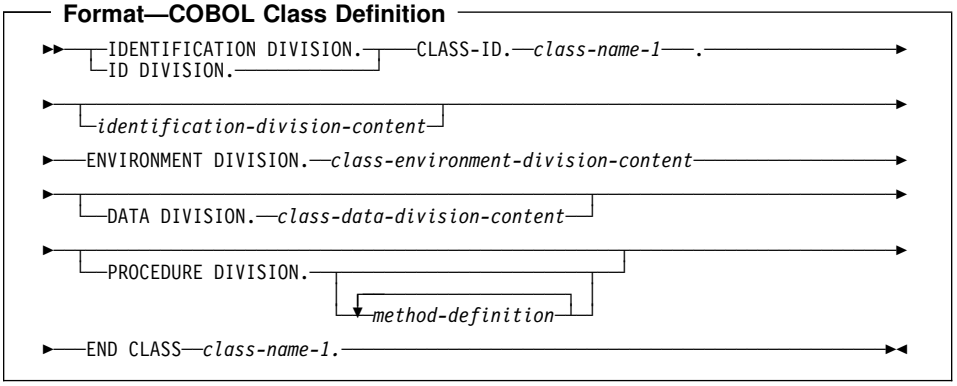
The end of a COBOL class definition is indicated by the END CLASS header.

Following is the format for the entries and statements that constitute a separately-compiled COBOL class definition.


© Copyright IBM Corp. 1991, 1996

## COBOL Class Definition

**Format—COBOL Class Definition**




**END CLASS**

Specifies the end of a class definition.

# COBOL Method Definition Structure

A COBOL method definition describes a method. You can only specify a method definition within a class definition.

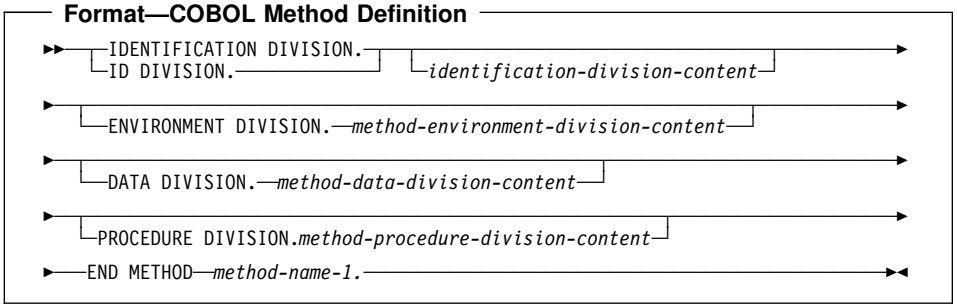
With the exception of the COPY and REPLACE statements and the END METHOD header, the statements, entries, paragraphs, and sections of a COBOL method definition are grouped into the following four divisions:

- Identification Division
- Environment Division (Input-Output section only)
- Data Division
- Procedure Division

The end of a COBOL method definition is indicated by the END METHOD header.

Following is the format for the entries and statements that constitute a separately-compiled COBOL method definition.

**Format—COBOL Method Definition**




**END METHOD**
Specifies the end of a method definition.

Methods defined in a class can access instance data (class Working-Storage Section data items) introduced in the same class but not instance data introduced by a parent class or metaclass. Therefore, instance data is always private to the class that introduces it.

Methods introduced in class-name-1 must have unique names within the class definition.


© Copyright IBM Corp. 1991, 1996

# Part 3. Identification Division


**Identification Division** . . . . . 72
PROGRAM-ID Paragraph . . . . . 75
x CLASS-ID Paragraph . . . . . 77
x METHOD-ID Paragraph . . . . . 79
Optional Paragraphs . . . . . 81


© Copyright IBM Corp. 1991, 1996

# Identification Division

The Identification Division must be the first division in every COBOL source program, class definition, and method definition. It names the program, class, or method, and can include the date the program, class, or method was written, the date of compilation, and other such documentary information. The Identification Division must begin with the words IDENTIFICATION DIVISION or ID DIVISION followed by a separator period.

**Program IDENTIFICATION DIVISION**

For a program, the first paragraph of the Identification Division must be the PROGRAM-ID paragraph.

The other paragraphs are optional, and as an IBM extension, can appear in any order.

**Class IDENTIFICATION DIVISION**

For a class, the first paragraph of the Identification Division must be the CLASS-ID paragraph.

The other paragraphs are optional, and can appear in any order.

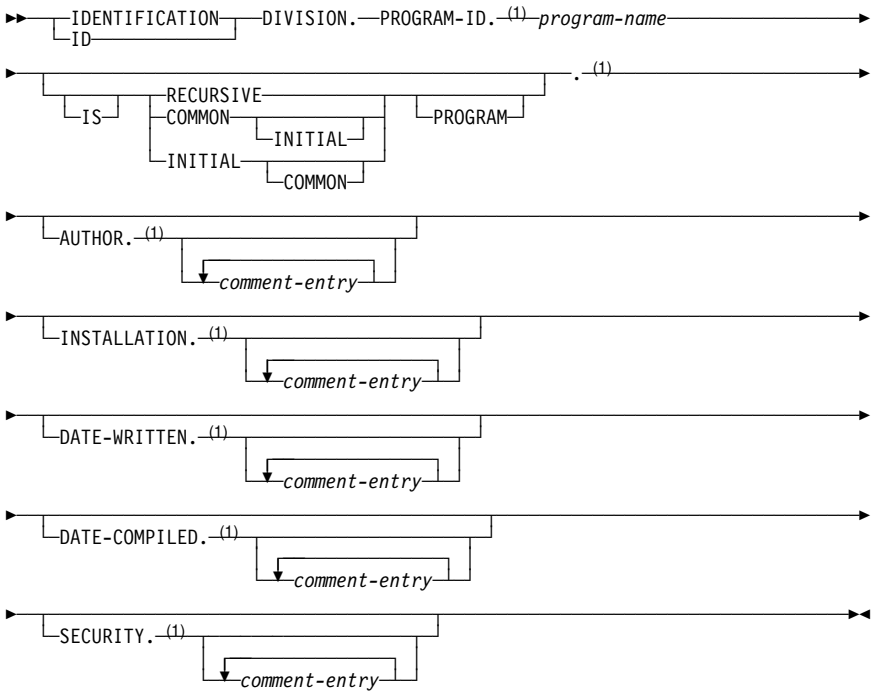
**Method IDENTIFICATION DIVISION**

For a method, the first paragraph of the Identification Division must be the METHOD-ID paragraph.

The other paragraphs are optional, and can appear in any order.


© Copyright IBM Corp. 1991, 1996

**Format—Program Identification Division**

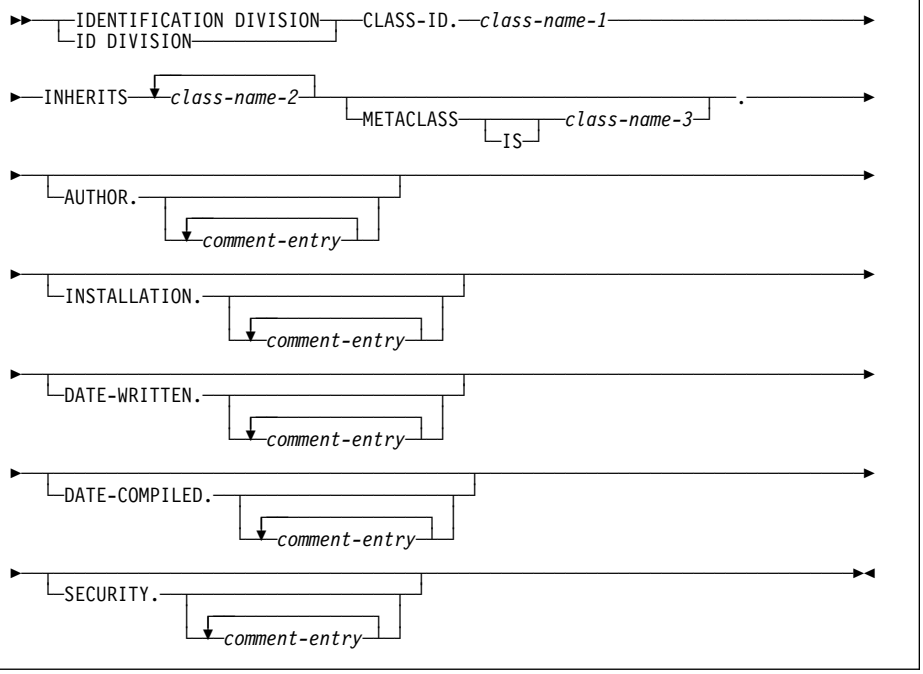



**Note:**

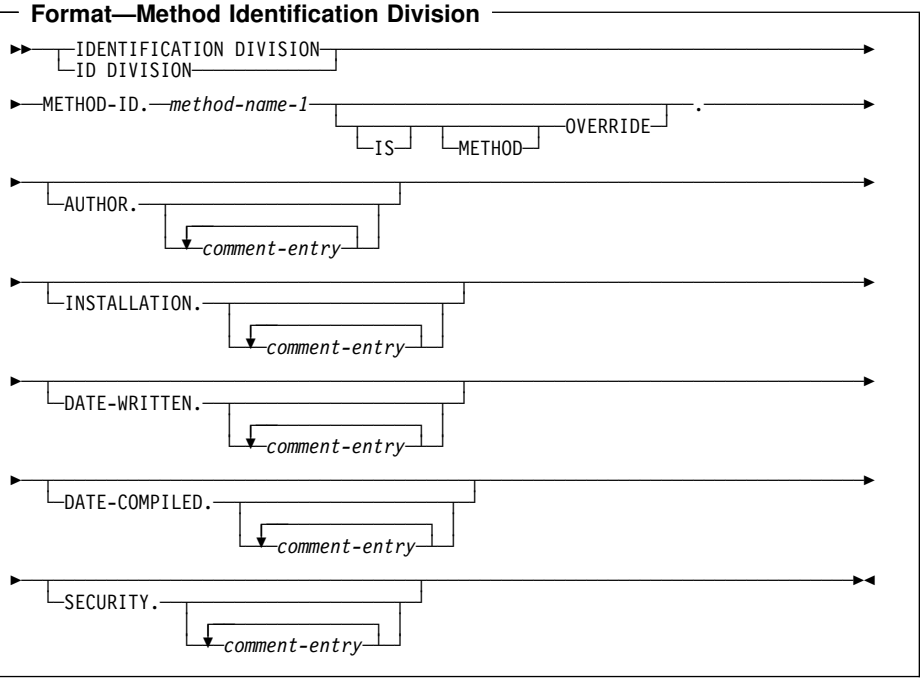
1 This separator period is optional as an IBM extension.

## Identification Division

**Format—Class Identification Division**




**Format—Method Identification Division**



## PROGRAM-ID Paragraph

The PROGRAM-ID paragraph specifies the name by which the program is known and assigns selected program attributes to that program. It is required and must be the first paragraph in the Identification Division.

**program-name**

x A user-defined word or nonnumeric literal that identifies your program. It must follow the following rules of formation, depending on the setting of the PGMNAME compiler option:

*Table 7. Formation Rules for Program Names Based on PGMNAME Compiler Option*

| PGMNAME Setting | Formation Rules | |
|---|---|---|
| | **MVS and VM** | **AIX, OS/2, and Windows** |
| PGMNAME (COMPAT) | The name can be up to 30 characters in length.<br>Only the hyphen, digit, and alphabetic characters are allowed in the name.<br>At least one character must alphabetic.<br>The hyphen cannot be used as the first or last character.<br>x If program-name is a nonnumeric literal, x (other than a figurative constant), it can x include the extension characters $, #, and x @ in the outermost program only. | Flagged with a warning message and treated as PGMNAME(UPPER). |
| PGMNAME (LONGUPPER) | If program-name is a user-defined word, it can be up to 30 characters in length.<br>x If program-name is a nonnumeric literal, it can be up to 160 characters in length. It cannot be x a figurative constant.<br>Only the hyphen, digit, and alphabetic characters are allowed in the name.<br>At least one character must alphabetic.<br>The hyphen cannot be used as the first or last character. | |
| x PGMNAME x (LONGMIXED) | Program-name must be specified as a literal. It cannot be a figurative constant.<br>x The name can be up to 160 characters in x length.<br>x Program-name can consist of any character x in the range X'41' to X'FE'. | Program-name must be specified as a literal. It cannot be a figurative constant.<br>x The name can be up to 160 characters in x length.<br>x Wherever alphabetic characters are allowed, x you can use multi-byte characters. |

x For information on the PGMNAME compiler option and how the compiler processes the x names, see the *IBM COBOL Programming Guide* for your platform.

x **RECURSIVE**

x An optional clause that allows COBOL programs to be recursively reentered.

x You can specify the RECURSIVE clause only on the outermost program of a com- x pilation unit. Recursive programs cannot contain nested subprograms.

If the RECURSIVE clause is specified, program-name-1 can be recursively reentered while a previous invocation is still active. If the RECURSIVE clause is not specified, an active program cannot be recursively reentered.

The Working-Storage Section of a recursive program defines storage that is statically allocated and initialized on the first entry to a program, and is available in a last-used state to any of the recursive invocations.

The Local-Storage Section of a recursive program (as well as a non-recursive program) defines storage that is automatically allocated, initialized, and deallocated on a per-invocation basis.

Internal file connectors corresponding to FDs in the File Section of a recursive program are statically allocated. The status of internal file connectors is part of the last-used state of a program that persists across invocations.

The following language elements are not supported in a recursive program:

- ALTER
- GO TO without a specified procedure name
- RERUN
- SEGMENTATION
- USE FOR DEBUGGING

**Note:** Methods are always recursive by default. The RECURSIVE clause cannot be specified on the METHOD-ID statement.

**COMMON**
Specifies that the program named by program-name is contained within another program, and it can be called from siblings of the common program and programs contained within them. The COMMON clause can be used only in nested programs. For more information on conventions for program names, see the *IBM COBOL Programming Guide* for your platform.

**INITIAL**
Specifies that when program-name is called, program-name and any programs contained within it are placed in their initial state.

A program is in the initial state:

- The first time the program is called in a run unit
- Every time the program is called, if it possesses the initial attribute
- The first time the program is called after the execution of a CANCEL statement referencing the program or a CANCEL statement referencing a program that directly or indirectly contains the program
- The first time the program is called after the execution of a CALL statement referencing a program that possesses the initial attribute, and that directly or indirectly contains the program.

When a program is in the initial state, the following occur:

- The program's internal data contained in the Working-Storage Section are initialized. If a VALUE clause is used in the description of the data item, the

data item is initialized to the defined value. If a VALUE clause is not associated with a data item, the initial value of the data item is undefined.

- Files with internal file connectors associated with the program are not in the open mode.
- The control mechanisms for all PERFORM statements contained in the program are set to their initial states.
- An altered GO TO statement contained in the program is set to its initial state.

For the rules governing non-unique program names, see "Rules for Program-Names" on page 66.

## CLASS-ID Paragraph

The CLASS-ID paragraph specifies the name by which the class is known and assigns selected attributes to that class. It is required and must be the first paragraph in a class Identification Division.

**class-name-1**
A user-defined word that identifies the class.

If you want to use more flexible naming conventions for class-name-1, specify class-name-1 in the REPOSITORY paragraph of the class definition. (This defines an external class name to identify the class outside of this class definition.)

**INHERITS**
A clause that defines class-name-1 to be a subclass (or derived class) of class-name-2 (the parent class). Class-name-1 cannot directly or indirectly inherit from class-name-1. A class name can only appear once in the INHERITS clause.

**class-name-2**
The name of a class inherited by class-name-1. If class-name-2 is repeated, multiple inheritance is present. You must specify class-name-2 in the REPOSITORY paragraph of the Configuration Section of the class definition.

**METACLASS**
A clause that identifies the metaclass for class-name-1. A metaclass is a special class whose instances are class objects. For more information on metaclasses, see the *IBM COBOL Programming Guide* for your platform.

Do not specify the METACLASS clause when defining a metaclass.

**Note:** The INHERITS and METACLASS clauses can appear in either order in the CLASS-ID paragraph.

**class-name-3**
The name of a metaclass that is responsible for creating and/or managing objects of the class being defined. You must specify class-name-3 in the REPOSITORY paragraph of the Configuration Section of the class definition.

## General Rules

Class-name-1, class-name-2, and class-name-3 must conform to the normal rules of formation for a COBOL user-defined word, as described in “COBOL Words with Single-Byte Characters” on page 5.

See “REPOSITORY Paragraph” on page 96 for details on:

- Class names mapping to CORBA compliant names
- Specification of external class-names with more flexible rules of formation

You can specify a sequence of class definitions and program definitions in a single COBOL source file, forming a batch compile.

## Inheritance

Every method available on instances of a class is also available on instances of any subclass directly or indirectly derived from it. A subclass can introduce new methods that do not exist in the parent (or ancestor) class or can override a method from the parent class. When a subclass overrides an existing method from the parent class, it defines a new implementation for that method, which replaces the inherited implementation.

The instance data of class-name-1 is a copy of the instance data from class-name-2 together with the data declared in the Working-Storage Section of class-name-1. Note however, instance data is always private to the class that introduces it.

The semantics of inheritance are defined by the IBM SOM. All classes must be derived directly or indirectly from the SOMObject class. All metaclasses must be derived directly or indirectly from SOMClass.

## Multiple Inheritance

Multiple inheritance is when more than one class name is specified on the INHERITS phrase. With multiple inheritance, a class might inherit the same methods and instance data from different parents (if each of these parents have a common ancestor). In this situation, (“diamond inheritance”) the subclass inherits only one set of method implementations and one copy of the instance data.

When a subclass inherits two methods with the same name, the two methods must comply to the following conformance rules:

- The number of formal parameters on the Procedure Division USING phrase must be the same for both methods.
- The presence or absence of the Procedure Division RETURNING phrase must be consistent for the two methods.

- Corresponding parameters in the Procedure Division USING and RETURNING phrases must satisfy the following:
  - If a formal parameter is a COBOL elementary data item not described with USAGE IS OBJECT REFERENCE, the corresponding parameter must have the same PICTURE, USAGE, SIGN, SYNCHRONIZED, JUSTIFIED, and BLANK WHEN ZERO clauses. Note that periods and commas can be interchanged if using the DECIMAL POINT IS COMMA clause, and the currency signs can differ.
  - If a formal parameter is a COBOL elementary data item described with USAGE IS OBJECT REFERENCE, the corresponding parameter must be defined with an identical USAGE clause or USAGE IS OBJECT REFERENCE clause.
  - For the purpose of conformance checking, a fixed-length group data item is considered to be equivalent to an elementary alphanumeric data item of the same length.

    A variable-length group conforms only to other variable-length groups that have the same maximum length.

## METHOD-ID Paragraph

The METHOD-ID paragraph specifies the name by which a method is known and assigns selected attributes to that method. It is required and must be the first paragraph in a method Identification Division.

**method-name-1**
A user-defined word or a nonnumeric literal that identifies the method.

The rules of formation for method-name-1 are as follows:

- If the method name is specified in the user-defined word format, then normal COBOL rules for a user-defined word apply.
- If the method name is specified as a nonnumeric literal, then:
  - The name can be up to 160 characters in length.
  - The characters used in the name must be uppercase or lowercase alphabetic, digit, hyphen, or underscore.
  - At least one character must be alphabetic.
  - Hyphen cannot be used as the first or last character.

**OVERRIDE**
A clause that allows a subclass to override an existing method implementation when it inherits a method from a parent class.

You must specify the OVERRIDE clause in the METHOD-ID paragraph, if method-name-1 is overriding a method with the same name that is inherited from a parent class.

Do not specify the OVERRIDE clause if the method is not inherited from an ancestor class, and is being introduced by the current class definition.

## General Rules

1. Method names that are defined for a class must be unique. (The set of methods "defined for a class" includes the methods introduced by the class definition and the methods inherited from parent classes.)

   **Note:** Method names that differ only in case are **not** considered unique. For example, naming one method “SAYHELLO” and another method “sayHELLO” is invalid.

2. Method names are processed by the compiler as follows:

   - Literal-format methods names are processed in a *case-sensitive* manner. However, when processing method resolution as part of INVOKE statements or method names that are specified as user-defined words, the compiler ignores any difference in case.
   - If necessary, the compiler translates method names to conform to CORBA requirements:
     - Hyphens are translated to zero
     - If the first character of the name is a digit, it is converted as follows:
       - 1 through 9 are changed to A through I
       - 0 is changed to J

3. If a method in class-name-1 overrides a method in class-name-2, these two methods must satisfy the following conformance rules:

   - The number of formal parameters on the Procedure Division USING phrase must be the same for both methods.
   - The presence or absence of the Procedure Division RETURNING phrase must be consistent on the two methods.
   - Corresponding parameters in the Procedure Division USING phrases must satisfy the following:
     - If a formal parameter is a COBOL elementary data item not described with USAGE IS OBJECT REFERENCE, then the corresponding parameter must have the same PICTURE, USAGE, SIGN, SYNCHRONIZED, JUSTIFIED, and BLANK WHEN ZERO clauses. Note that periods and commas can be interchanged if using the DECIMAL POINT IS COMMA clause, and the currency signs can differ.
     - If a formal parameter is a COBOL elementary data item described with USAGE IS OBJECT REFERENCE, then the corresponding parameter must be defined with an identical USAGE IS OBJECT REFERENCE clause.
     - BY VALUE and BY REFERENCE specifications must be consistent.

- The identifiers specified on the Procedure Division RETURNING phrases must satisfy the following:
  - If one of the identifiers is a COBOL elementary data item not described with USAGE IS OBJECT REFERENCE, then the corresponding identifier must have the same PICTURE, USAGE, SIGN, SYNCHRONIZED, JUSTIFIED, and BLANK WHEN ZERO clauses. Note that periods and commas can be interchanged if using the DECIMAL POINT IS COMMA clause, and the currency signs can differ.
  - If the class-name-2 Procedure Division RETURNING identifier is a universal object reference, the class-name-1 Procedure Division RETURNING identifier must be an object reference (either a universal object reference or an object reference typed to a specific class).

    Universal object references are described with USAGE OBJECT REFERENCE and typed object references are described with USAGE OBJECT REFERENCE class-name.
  - If the class-name-2 Procedure Division RETURNING identifier is an object reference typed to a specific class, the class-name-1 Procedure Division RETURNING identifier must be an object reference typed to the same class or a derived class.
- For the purpose of conformance checking, a fixed-length group data item is considered to be equivalent to an elementary alphanumeric data item of the same length.

  A variable-length group conforms only to other variable-length groups that have the same maximum length.

## Optional Paragraphs

These optional paragraphs in the Identification Division can be omitted:

**AUTHOR**
Name of the author of the program.

**INSTALLATION**
Name of the company or location.

**DATE-WRITTEN**
Date the program was written.

**DATE-COMPILED**
Date the program was compiled.

**SECURITY**
Level of confidentiality of the program.

The **comment-entry** in any of the optional paragraphs can be any combination of characters from the character set of the computer. The comment-entry is written in Area B on one or more lines.

## Optional Paragraphs

The paragraph name DATE-COMPILED and any comment-entry associated with it appear in the output program listing with the current date inserted:

```
DATE-COMPILED. 04/27/95.
```

Comment-entries serve only as documentation; they do not affect the meaning of the program. A hyphen in the indicator area (column 7) is not permitted in comment-entries.

x WRKSTN Under AIX, OS/2, and Windows, you can include multi-byte as well as single-
x byte characters in an EUC or DBCS code page in comment entries in the Identification
x Division of your program. Multiple lines are allowed in a comment-entry containing
x multi-byte characters.

x MVS/VM Under MVS and VM, you can include DBCS character strings as comment-
x entries in the Identification Division of your program. Multiple lines are allowed in a
x comment-entry containing DBCS strings.

x A DBCS string must be preceded by a shift-out control character and followed by a
x shift-in control character. For example:

```
x AUTHOR.       <.A.U.T.H.O.R.-.N.A.M.E>, XYZ CORPORATION
x DATE-WRITTEN. <.D.A.T.E>
```

x When using DBCS characters in a comment-entry contained on multiple lines, shift-out
x and shift-in characters must be paired on a line.

x DBCS strings are described under “Character-Strings” on page 5.

# Part 4. Environment Division


**Configuration Section** . . . . . 84
SOURCE-COMPUTER Paragraph . . . . . 85
OBJECT-COMPUTER Paragraph . . . . . 86
SPECIAL-NAMES Paragraph . . . . . 88
ALPHABET Clause . . . . . 91
SYMBOLIC CHARACTERS Clause . . . . . 93
CLASS Clause . . . . . 94
CURRENCY SIGN Clause . . . . . 95
x REPOSITORY Paragraph . . . . . 96

**Input-Output Section** . . . . . 98
FILE-CONTROL Paragraph . . . . . 99
SELECT Clause . . . . . 103
ASSIGN Clause . . . . . 103
RESERVE Clause . . . . . 107
ORGANIZATION Clause . . . . . 108
PADDING CHARACTER Clause . . . . . 111
RECORD DELIMITER Clause . . . . . 111
ACCESS MODE Clause . . . . . 112
RECORD KEY Clause . . . . . 114
ALTERNATE RECORD KEY Clause . . . . . 115
RELATIVE KEY Clause . . . . . 116
x PASSWORD Clause . . . . . 116
x LOCK MODE Clause . . . . . 117
FILE STATUS Clause . . . . . 119
I-O-CONTROL Paragraph . . . . . 120
RERUN Clause . . . . . 122
SAME AREA Clause . . . . . 123
SAME RECORD AREA Clause . . . . . 124
SAME SORT AREA Clause . . . . . 125
SAME SORT-MERGE AREA Clause . . . . . 125
MULTIPLE FILE TAPE Clause . . . . . 126
APPLY WRITE-ONLY Clause . . . . . 126


© Copyright IBM Corp. 1991, 1996

# Configuration Section

The Configuration Section is an optional section for programs and classes, which can describe the computer environment on which the program is compiled and executed.

**Program Configuration Section**

The Configuration Section can be specified only in the ENVIRONMENT DIVISION of the outermost program of a COBOL source program.

You should not specify the Configuration Section in a program that is contained within another program. The entries specified in the Configuration Section of a program apply to any program contained within that program.

**Class Configuration Section**

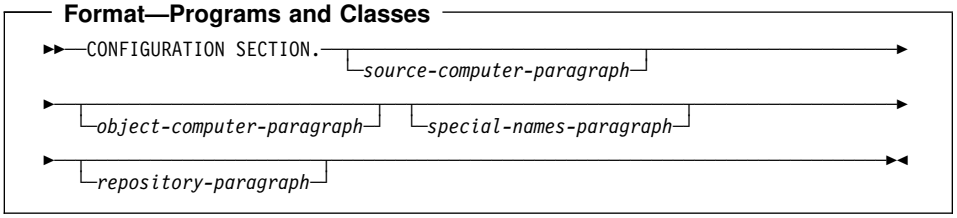
Specify the Configuration Section only in the ENVIRONMENT DIVISION of the outermost program of a class definition.

Entries in a class Configuration Section apply to the entire class definition, including all methods introduced by that class.

**Method Configuration Section**

The Configuration Section is **not** valid for method definitions.

**Format—Programs and Classes**




The Configuration Section can:

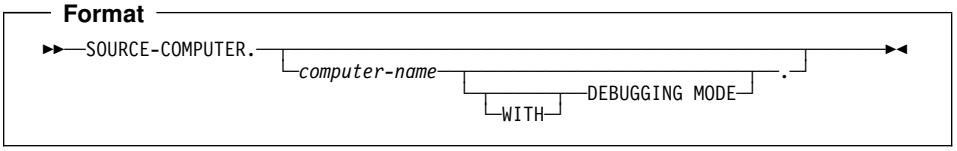
- Relate IBM-defined environment-names to user-defined mnemonic names
- Specify the collating sequence
- Specify a substitution for the currency sign
- Exchange the functions of the comma and the period in PICTURE clauses and numeric literals
- Relate alphabet-names to character sets or collating sequences
- Specify symbolic-characters
- Relate class names to sets of characters
- Relate object-oriented class names to the class names in the SOM interface repository


© Copyright IBM Corp. 1991, 1996

## SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph describes the computer on which the source program is to be compiled.

**Format**




**computer-name**
A system-name. For example:

```
IBM-370
```

**WITH DEBUGGING MODE**
Activates a compile-time switch for debugging lines written in the source program.

A debugging line is a statement that is compiled only when the compile-time switch is activated. Debugging lines allow you, for example, to check the value of a data-name at certain points in a procedure.

To specify a debugging line in your program, code a 'D' in column 7 (indicator area). You can include successive debugging lines, but each must have a 'D' in column 7 and you cannot break character strings across lines.

All your debugging lines must be written so that the program is syntactically correct, whether the debugging lines are compiled or treated as comments.

The presence or absence of the DEBUGGING MODE clause is logically determined after all COPY and REPLACE statements have been processed.

You can code debugging lines in the Environment (after the OBJECT-COMPUTER paragraph), Data, or Procedure Divisions.

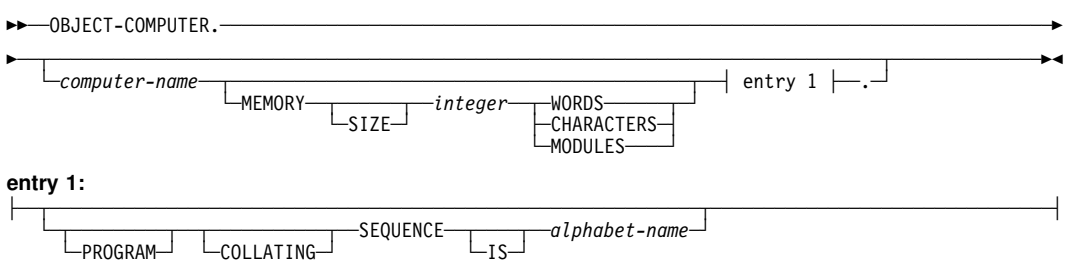
If a debugging line contains only spaces in Area A and in Area B, it is treated the same as a blank line.

Except for the WITH DEBUGGING MODE clause, the SOURCE-COMPUTER paragraph is syntax checked, but has no effect on the execution of the program.

## OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph specifies the system for which the object program is designated.

**Format**




**computer-name**
A system-name. For example:

```
IBM-370
```

**MEMORY SIZE**
The amount of main storage needed to run the object program. The MEMORY SIZE clause is syntax checked, but it has no effect on the execution of the program.

**integer**
Expressed in words, characters, or modules.

**PROGRAM COLLATING SEQUENCE IS**
The collating sequence used in this program is the collating sequence associated with the specified alphabet-name.

The collating sequence pertains to this program and any programs it might contain.

**alphabet-name**
The collating sequence.

PROGRAM COLLATING SEQUENCE determines the truth value of the following nonnumeric comparisons:

- Those explicitly specified in relation conditions
- Those explicitly specified in condition-name conditions

The PROGRAM COLLATING SEQUENCE clause also applies to any nonnumeric merge or sort keys, unless the COLLATING SEQUENCE phrase is specified in the MERGE or SORT statement.

x MVS/VM Under MVS and VM, the PROGRAM COLLATING SEQUENCE clause is not
x applied to the DBCS character set.

x WRKSTN Under AIX, OS/2, and Windows, the PROGRAM COLLATING SEQUENCE
x clause is not allowed if the code page in effect is a DBCS or EUC code page.

When the PROGRAM COLLATING SEQUENCE clause is omitted:

- MVS/VM Under MVS and VM, the EBCDIC collating sequence is used. (See Appendix B, "EBCDIC and ASCII Collating Sequences" on page 522.)
- WRKSTN Under AIX, OS2, and Windows, the COLLSEQ compiler option indicates the collating sequence used. For example, if COLLSEQ(EBCDIC) is specified and the PROGRAM COLLATING SEQUENCE is not specified (or is NATIVE), the EBCDIC collating sequence is applied.

**SEGMENT-LIMIT IS**

Certain permanent segments can be overlaid by independent segments while still retaining the logical properties of **fixed portion** segments. (Fixed portion segments are made up of fixed permanent and fixed overlayable segments.)

**Priority-number**

An integer ranging from 1 through 49.

When SEGMENT-LIMIT is specified:

- A fixed **permanent** segment is one with a priority-number less than the priority-number specified.
- A fixed **overlayable** segment is one with a priority-number ranging from that specified through 49, inclusive.

For example, if SEGMENT-LIMIT IS 25 is specified:

- Sections with priority-numbers 0 through 24 are fixed **permanent** segments.
- Sections with priority-numbers 25 through 49 are fixed **overlayable** segments.

When SEGMENT-LIMIT is omitted, all sections with priority-numbers 0 through 49 are fixed permanent segments.

Except for the PROGRAM COLLATING SEQUENCE clause, the OBJECT-COMPUTER paragraph is syntax checked, but it has no effect on the execution of the program.

## SPECIAL-NAMES Paragraph

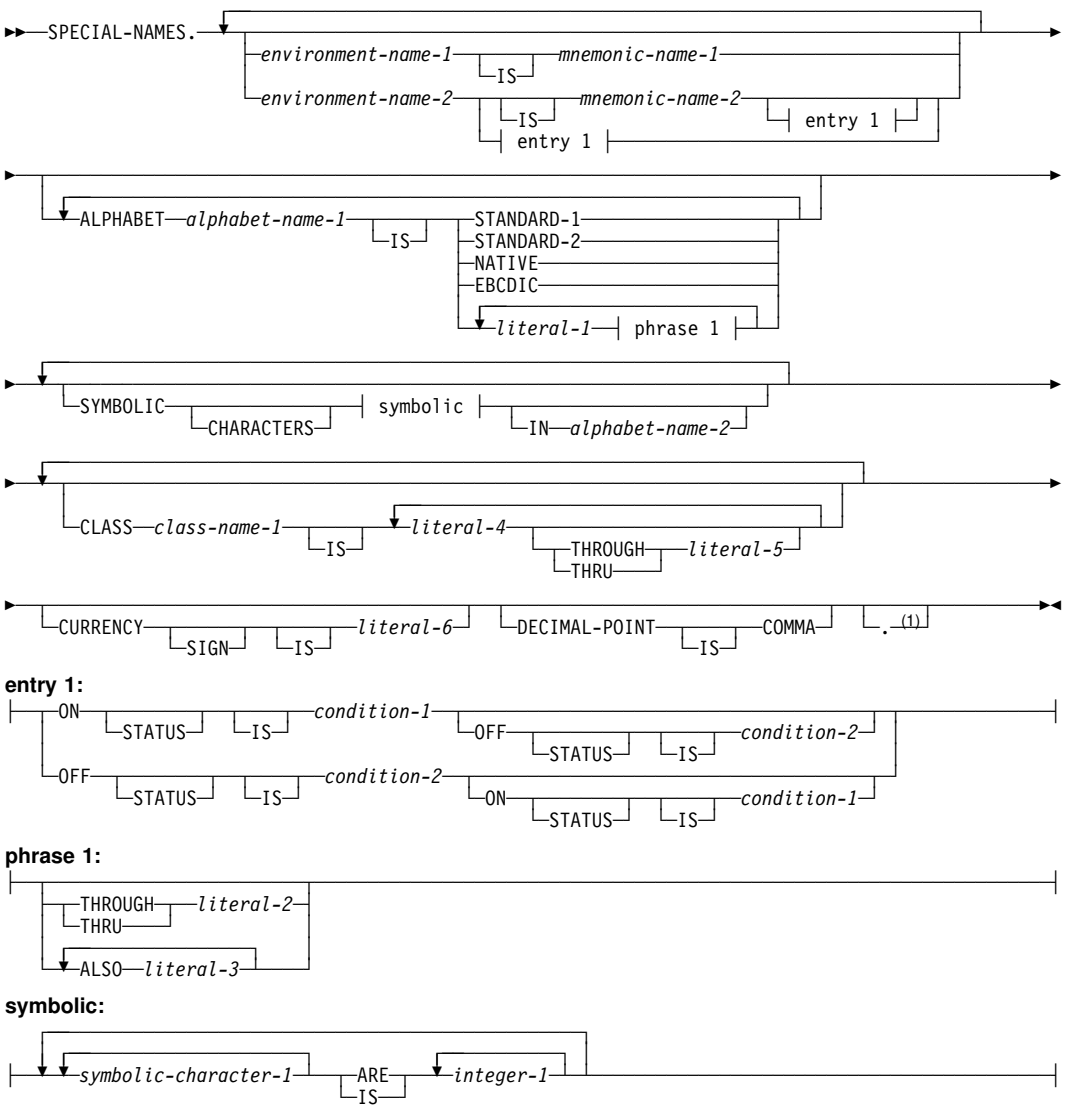
The SPECIAL-NAMES paragraph:

- Relates IBM-specified environment-names to user-defined mnemonic-names
- Relates alphabetic-names to character sets or collating sequences
- Specifies symbolic characters
- Relates class names to sets of characters
- Specifies a substitute character for the currency sign
- Specifies that the functions of the comma and decimal point are to be interchanged in PICTURE clauses and numeric literals.

x **Note:** The clauses in the SPECIAL-NAMES paragraph can appear in any order.

## SPECIAL-NAMES Paragraph

**Format**




**Note:**

x [1] This separator period must be used if any of the optional clauses are selected.

**environment-name-1**
System devices or standard system actions taken by the compiler.

Valid specifications for environment-name-1 are:

*Table 8. Meanings of Environment Names*

| Environment Name-1 | Meaning | Allowed In |
|---|---|---|
| SYSIN<br>SYSIPT | System logical input unit | ACCEPT |
| SYSOUT<br>SYSLIST<br>SYSLST | System logical output unit | DISPLAY |
| SYSPUNCH<br>SYSPCH | System punch device | DISPLAY |
| CONSOLE | Console | ACCEPT and DISPLAY |
| C01 - C12 | Skip to channel 1 through 12, respectively | WRITE ADVANCING (On AIX, OS/2, and Windows, with C01 - C12, one line is advanced.) |
| CSP | Suppress spacing | WRITE ADVANCING |
| S01-S05 | Pocket select 1-5 on punch devices | WRITE ADVANCING (On AIX, OS/2, and Windows, with S01 - S05, one line is advanced.) |
| AFP-5A | Advanced function printing | WRITE ADVANCING |

**environment-name-2**
A 1-byte User Programmable Status Indicator (UPSI) switch. Valid specifications for environment-name-2 are UPSI-0 through UPSI-7.

**mnemonic-name-1, mnemonic-name-2**
Mnemonic-name-1 and mnemonic-name-2 follow the rules of formation for user-defined names. Mnemonic-name-1 can be used in ACCEPT, DISPLAY, and WRITE statements. Mnemonic-name-2 can be referenced only in the SET statement. Mnemonic-name-2 can qualify cond-1 or cond-2 names.

Mnemonic-names and environment-names need not be unique. If you choose a mnemonic-name that is also an environment-name, its definition as a mnemonic-name will take precedence over its definition as an environment-name.

**ON STATUS IS, OFF STATUS IS**
UPSI switches process special conditions within a program, such as year-beginning or year-ending processing. For example, at the beginning of the Procedure Division, an UPSI switch can be tested; if it is ON, the special branch is taken. (See “Switch-Status Condition” on page 238.)

**cond-1, cond-2**

Condition-names follow the rules for user-defined names. At least one character must be alphabetic. The value associated with the condition-name is considered to be alphanumeric. A condition-name can be associated with the on status and/or off status of each UPSI switch specified.

In the Procedure Division, the UPSI switch status is tested through the associated condition-name. Each condition-name is the equivalent of a level-88 item; the associated mnemonic-name, if specified, is considered the conditional variable and can be used for qualification.

Condition-names specified in a containing program's SPECIAL-NAMES paragraph can be referenced from any contained program.

## ALPHABET Clause

**ALPHABET alphabet-name-1 IS**

Provides a means of relating an alphabet-name to a specified character code set or collating sequence.

It specifies a **collating sequence** when used in either:

- The PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph
- The COLLATING SEQUENCE phrase of the SORT or MERGE statement

It specifies a **character code set** when specified in either:

- The FD entry CODE-SET clause
- The SYMBOLIC CHARACTERS clause

x WRKSTN Under AIX, OS/2, and Windows, you cannot specify the ALPHABET
x clause if the code page in effect is a DBCS or EUC code page. For details, see
x the *IBM COBOL Programming Guide* for your platform.

**STANDARD-1**

MVS/VM Under MVS and VM, specifies the ASCII character set.

WRKSTN Under AIX, OS/2, and Windows, specifies that the collating sequence is based on the binary code values of the characters, ignoring the locale setting.

**STANDARD-2**

MVS/VM Under MVS and VM, specifies the International Reference Version of the ISO 7-bit code defined in International Standard 646, 7-bit Coded Character Set for Information Processing Interchange.

WRKSTN Under AIX, OS/2, and Windows, specifies that the collating sequence is based on the binary code values of the characters, ignoring the locale setting.

## ALPHABET Clause

**NATIVE**
Specifies the native character code set. If the alphabet-name clause is omitted:

MVS/VM Under MVS and VM, EBCDIC is assumed.

WRKSTN Under AIX, OS/2, and Windows, if the COLLSEQ(EBCDIC) compiler option is specified, then the EBCDIC collating sequence is used. Otherwise, DBCS and EUC data comparisons are always preformed with the locale sensitive collating sequence. For information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 543.

**EBCDIC**
Specifies the EBCDIC character set.

**literal-1**
**literal-2**
**literal-3**
Specifies that the collating sequence is to be determined by the program, according to the following rules:

- The order in which literals appear specifies the ordinal number, in ascending sequence, of the character(s) in this collating sequence.
- Each numeric literal specified must be an unsigned integer.
- Each numeric literal must have a value that corresponds to a valid ordinal position within the collating sequence in effect.

  Appendix B, "EBCDIC and ASCII Collating Sequences" on page 522, lists the ordinal number for characters in the EBCDIC and ASCII collating sequences.
- Each character in a nonnumeric literal represents that actual character in the character set. (If the nonnumeric literal contains more than one character, each character, starting with the leftmost, is assigned a successively ascending position within this collating sequence.)
- Any characters that are not explicitly specified assume positions in this collating sequence higher than any of the explicitly specified characters. The relative order within the set of these unspecified characters within the character set remains unchanged.
- Within one alphabet-name clause, a given character must not be specified more than once.
- Each nonnumeric literal associated with a THROUGH or ALSO phrase must be 1 character in length.
- When the THROUGH phrase is specified, the contiguous characters in the native character set beginning with the character specified by literal-1 and ending with the character specified by literal-2 are assigned successively ascending positions in this collating sequence. This sequence can be either ascending or descending within the original native character set.

That is, if "Z" THROUGH "A" is specified, the ascending values, left-to-right, for the uppercase letters are:

```
ZYXWVUTSRQPONMLKJIHGFEDCBA
```

- When the ALSO phrase is specified, the characters specified as literal-1, literal-3, etc., are assigned to the same position in this collating sequence. For example, if you specify:

  ```
  "D" ALSO "N" ALSO "%"
  ```

  the characters D, N, and % are all considered to be in the same position in the collating sequence.
- When the ALSO phrase is specified and alphabet-name-1 is referenced in a SYMBOLIC CHARACTERS clause, only literal-1 is used to represent the character in the character set.
- The character having the **highest** ordinal position in this collating sequence is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position, because of specification of the ALSO phrase, the last character specified (or defaulted to when any characters are not explicitly specified) is considered to be the HIGH-VALUE character for procedural statements such as DISPLAY, or as the sending field in a MOVE statement. (If all characters and the ALSO phrase example given above were specified as the high-order characters of this collating sequence, the HIGH-VALUE character would be %.)
- The character having the **lowest** ordinal position in this collating sequence is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position, because of specification of the ALSO phrase, the first character specified is the LOW-VALUE character. (If the ALSO phrase example given above were specified as the low-order characters of the collating sequence, the LOW-VALUE character would be D.)

When **literal-1**, **literal-2**, or **literal-3** is specified, the alphabet-name must **not** be referred to in a CODE-SET clause (see “CODE-SET Clause” on page 156).

**Literal-1**, **literal-2**, and **literal-3** must not specify a symbolic-character figurative constant.

x Floating-point literals cannot be used in a user-specified collating sequence.

x DBCS literals cannot be used in a user-specified collating sequence.

## SYMBOLIC CHARACTERS Clause

**SYMBOLIC CHARACTERS symbolic-character-1**

Provides a means of specifying one or more symbolic characters.
Symbolic-character-1 is a user-defined word and must contain at least one alphabetic character. The same symbolic-character can appear only once in a SYMBOLIC CHARACTERS clause.

## CLASS Clause

x MVS/VM Under MVS and VM, the symbolic character can be a DBCS user-
x defined word.

x WRKSTN Under AIX, OS/2, and Windows, you cannot use the SYMBOLIC CHAR-
x ACTER phrase if the code page is DBCS or EUC.

The internal representation of symbolic-character-1 is the internal representation of the character that is represented in the specified character set. The following rules apply:

- The relationship between each symbolic-character-1 and the corresponding integer-1 is by their 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 forth.
- There must be a one-to-one correspondence between occurrences of symbolic-character-1 and occurrences of integer-1 in a SYMBOLIC CHARACTERS clause.
- If the IN phrase is specified, integer-1 specifies the ordinal position of the character that is represented in the character set named by alphabet-name-2. This ordinal position must exist.
- If the IN phrase is not specified, symbolic-character-1 represents the character whose ordinal position in the native character set is specified by integer-1.

  **Note:** Ordinal positions are numbered starting from 1.

## CLASS Clause

x WRKSTN Under AIX, OS/2, and Windows, you cannot specify the CLASS clause if the
x code page in effect is a DBCS or EUC code page.

**CLASS class-name-1 IS**
Provides a means for relating a name to the specified set of characters listed in that clause. Class-name can be referenced only in a class condition. The characters specified by the values of the literals in this clause define the exclusive set of characters of which this class-name consists.

x MVS/VM Under MVS and VM, the class-name in the CLASS clause can be a
x DBCS user-defined word.

**literal-4, literal-5**
If numeric, must be unsigned integers and must have a value that is greater than or equal to 1 and less than or equal to the number of characters in the alphabet specified. Each number corresponds to the ordinal position of each character in
x the EBCDIC or ASCII collating series. Cannot be specified as floating-point literals
x or as DBCS literals.

If nonnumeric, the literal is the actual EBCDIC or ASCII character. **Literal-4** and **literal-5** must not specify a symbolic-character figurative constant. If the value of the nonnumeric literal contains multiple characters, each character in the literal is included in the set of characters identified by class-name.

If the nonnumeric literal is associated with a THROUGH phrase, it must be one character in length.

**THROUGH, THRU**
THROUGH and THRU are equivalent. If THROUGH is specified, class-name includes those characters beginning with the value of literal-4 and ending with the value of literal-5. In addition, the characters specified by a THROUGH phrase can specify characters in either ascending or descending order.

## CURRENCY SIGN Clause

**CURRENCY SIGN IS**
Currency symbol in the PICTURE clause.

**literal-6**
Must be a one-character, nonnumeric literal, and must **not** be any of the following:

- Digits zero (0) through nine (9)
- Uppercase alphabetic characters A B C D P R S V X Z
- Lowercase alphabetic characters a through z
- The space
- Special characters * + - / , . ; ( ) = '
- A figurative constant
- The uppercase alphabetic character G. If the COBOL program defines a DBCS item with the PICTURE symbol G, then the PICTURE clause will be invalid for that DBCS item because the symbol G is considered to be a currency symbol in the PICTURE clause
- The uppercase alphabetic character N. If the COBOL program defines a DBCS with the PICTURE symbol N, then the PICTURE clause will be invalid for that DBCS item because the symbol N is considered to be a currency symbol in the PICTURE clause
- The uppercase alphabetic character E. If the COBOL program defines an external floating-point item, then the PICTURE clause will be invalid for the external floating-point item because the symbol E is considered to be a currency symbol in the PICTURE clause
- Under MVS and VM, the non-printable characters with hex values of X'20' and X'21'

The CURRENCY SIGN clause literal takes precedence over the default currency sign established by the CURRENCY compiler option.

When the NOCURRENCY option is in effect and the CURRENCY SIGN clause is omitted, only the dollar sign ($) can be used as the PICTURE symbol for the currency sign.

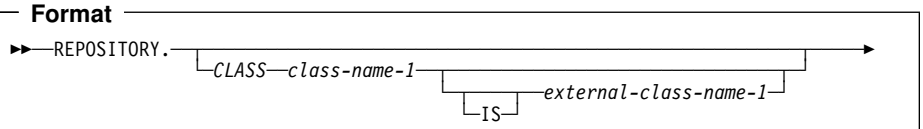
For more information about the CURRENCY/NOCURRENCY compiler options see the *IBM COBOL Programming Guide* for your platform.

**DECIMAL-POINT IS COMMA**
Exchanges the functions of the period and the comma in PICTURE character strings and in numeric literals.

## REPOSITORY Paragraph

The REPOSITORY paragraph defines the names of the classes that you can use in a class definition or program. Optionally, the REPOSITORY paragraph defines associations between class-names and external class-names.

**Format**




**class-name-1**
A user-defined word that identifies the class.

**external-class-name-1**
A name that enables a COBOL program to define or access classes with names that are defined using CORBA rules of formation. (Class names defined using CORBA rules of formation might not be expressible as a COBOL user-defined word, such as the case-sensitive SOM class names (SOMObject for example), or a class implemented in C with a name containing underscores.)

You must specify *external-class-name-1* as a nonnumeric literal, conforming to the following rules of formation:

- The name must not be a figurative constant.
- The name can be up to 160 characters in length.
- The characters used in the name must be uppercase or lowercase alphabetic, digit, or underscore.
- The leading character must be alphabetic.

## General Rules

1. All class names (whether referenced in a program, class definition, or method introduced by the class) must have an entry in the REPOSITORY paragraph. (You do not have to put the name of the class you are defining in the REPOSITORY paragraph. Note, if you don't, the class name is stored in all uppercase in the SOM repository.)

   You can only specify a class name once in a given REPOSITORY paragraph.

2. Entries in a class REPOSITORY paragraph apply to the entire class definition, including all methods introduced by that class. Entries in a program REPOSITORY paragraph apply globally to all nested programs contained within the program.

## Identifying and Referencing the Class

The external class-name is used to identify and reference the class outside of the source file containing the class definition (for example, to identify the entry for the class in the SOM Interface Repository). The external class-name is determined by using the contents of either *external-class-name-1* or *class-name-1* (as specified in the REPOSITORY paragraph of a class), as described below:

1. *external-class-name-1*—is used directly, without translations. The external class-names are processed in a case-sensitive manner.
2. *class-name-1*—is used if *external-class-name-1* is not specified. To create a CORBA-compliant external name that identifies the class, *class-name-1* is processed as follows:
   - The name is converted to uppercase.
   - Hyphens are translated to zero.
   - If the first character of the name is a digit, it is converted as follows:
     - 1 though 9 are changed to A through I
     - 0 is changed to J

# Input-Output Section

The Input-Output Section of the Environment Division contains two paragraphs:

- FILE-CONTROL paragraph
- I-O-CONTROL paragraph

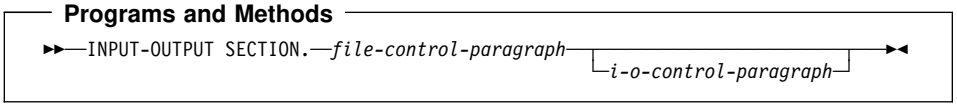
The exact contents of the Input-Output Section depend on the file organization and access methods used. See "ORGANIZATION Clause" on page 108 and "ACCESS MODE Clause" on page 112.

**Program Input-Output Section**
The same rules apply to program and method I-O Sections.

x **Class Input-Output Section**
x The Input-Output Section is **not** valid for class definitions.

x **Method Input-Output Section**
x The same rules apply to program and method I-O Sections.

**Programs and Methods**




**FILE-CONTROL paragraph-name**
The key word FILE-CONTROL can appear only once, at the beginning of the FILE-CONTROL paragraph. It must begin in Area A, and be followed by a separator period.

**file-control-paragraph**
Names the files and associates them with the external data sets.

Must begin in Area B with a SELECT clause. It must end with a separator period. See "FILE-CONTROL Paragraph" on page 99.

**I-O-CONTROL paragraph-name**
Specifies information needed for efficient transmission of data between the external data set and the COBOL program.

**input-output-control-paragraph**
The series of entries must end with a separator period. See "I-O-CONTROL Paragraph" on page 120.


© Copyright IBM Corp. 1991, 1996

## FILE-CONTROL Paragraph

The FILE-CONTROL paragraph associates each file in the COBOL program with an external data set, and specifies file organization, access mode, and other information.

The following are the formats for the FILE-CONTROL paragraph:

- Sequential file entries
- Indexed file entries
- Relative file entries
- x Line sequential file entries (Workstation only)

Table 9 lists the different type of files available to mainframe and workstation COBOL programs.

*Table 9. Types of Files*

| File Organization | Access Method MVS and VM | File Systems | | |
|---|---|---|---|---|
| | | AIX | OS/2 | Windows |
| Sequential | QSAM, VSAM | VSAM[1], STL | VSAM, Btrieve, STL | VSAM, STL |
| Relative | VSAM | VSAM[1], STL | VSAM, Btrieve, STL | VSAM, STL |
| Indexed | VSAM | VSAM[1], STL | VSAM, Btrieve, STL | VSAM, STL |
| Line Sequential | n/a | Native | Native | Native |

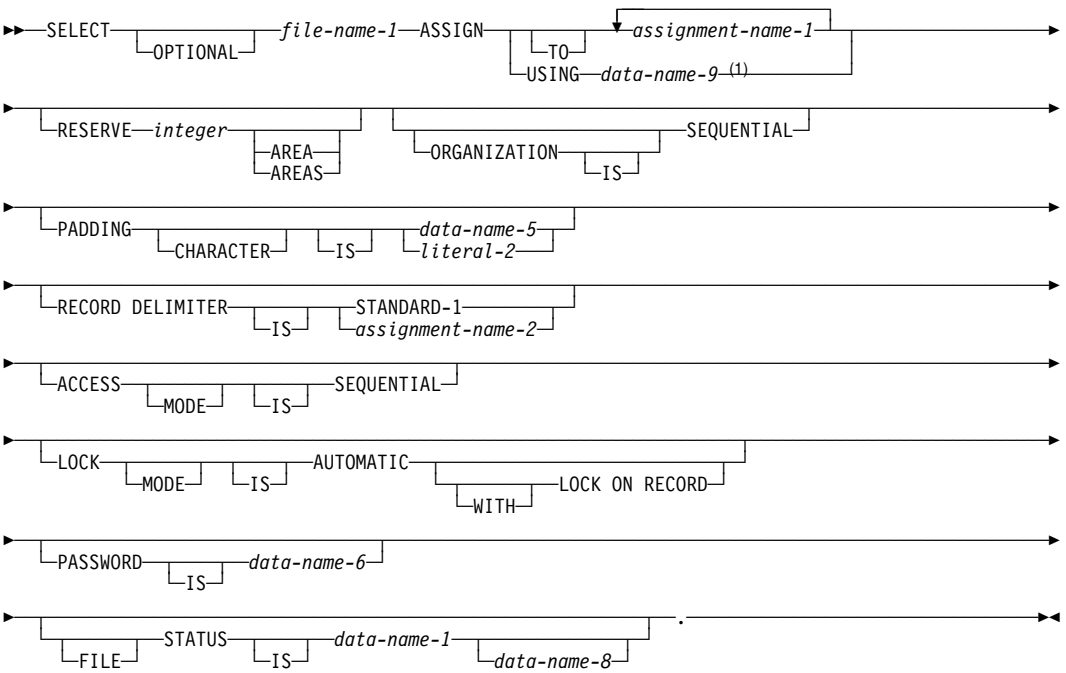
**Note:** [1] On AIX, you access the SFS file system through VSAM.

The FILE-CONTROL paragraph begins with the word "FILE-CONTROL", followed by a separator period. It must contain one and only one entry for each file described in an FD or SD entry in the Data Division. Within each entry, the SELECT clause must appear first. The other clauses can appear in any order.

x MVS/VM Under MVS and VM, there is one exception to the rule about order. For
x indexed files, the PASSWORD clause, if specified, must immediately follow the
x RECORD KEY or ALTERNATE RECORD KEY data-name with which it is associated.
x

## FILE-CONTROL Paragraph

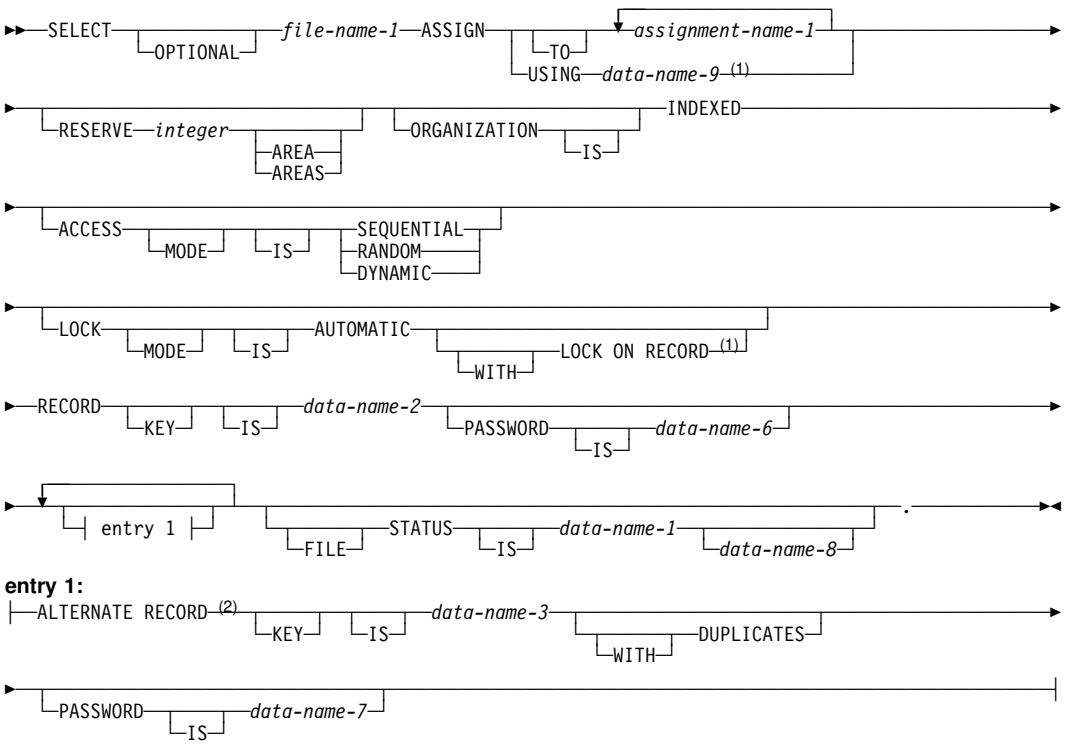
**Format 1—Sequential-File-Control-Entries**




**Note:**

1 The USING data-name phrase of the ASSIGN clause and the LOCK clause are only valid under AIX, OS/2, and Windows.

**Format 2—Indexed-File-Control-Entries**

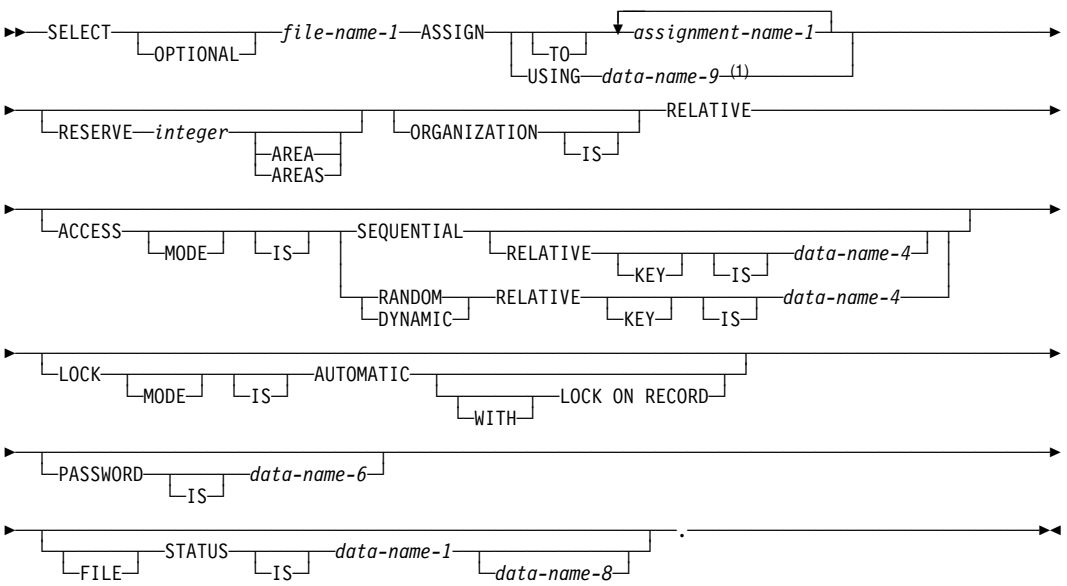



**Notes:**

1 The USING data-name phrase of the ASSIGN clause and the LOCK clause are only valid under AIX, OS/2, and Windows.

2 RECORD is optional as an IBM extension.

**Format 3—Relative-File-Control-Entries**




**Note:**

1 The USING data-name phrase of the ASSIGN clause and the LOCK clause are only valid under AIX, OS/2, and Windows.

## SELECT Clause

The SELECT clause chooses a file in the COBOL program to be associated with an external data set.

**SELECT OPTIONAL**
Can be specified only for files opened in the input, I-O, or extend mode. You must specify SELECT OPTIONAL for such input files that are not necessarily present each time the object program is executed. For more information, see the *IBM COBOL Programming Guide* for your platform.

**file-name**
Must be identified by an FD or SD entry in the Data Division. A file-name must conform to the rules for a COBOL user-defined name, must contain at least one alphabetic character, and must be unique within this program.

When file-name specifies a sort or a merge file, only the ASSIGN clause can follow the SELECT clause.

If the file connector referenced by file-name-1 is an external file connector, all file control entries in the run unit that reference this file connector must have the same specification for the OPTIONAL phrase.

## ASSIGN Clause

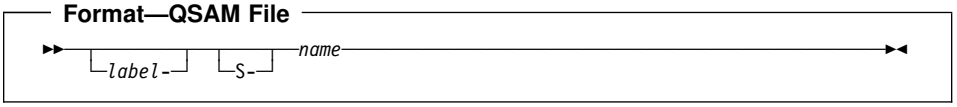
The ASSIGN clause associates the program's name for a file with the external name for the actual data file.

## MVS and VM Syntax

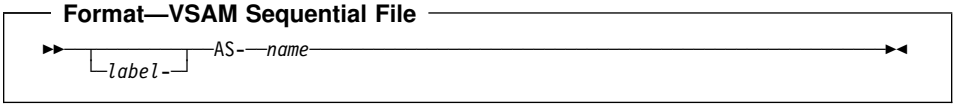
**assignment-name-1**
Can be specified as a user-defined word or a nonnumeric literal. Any assignment-name after the first is syntax checked, but it has no effect on the execution of the program.

Assignment-name-1 has the following formats:

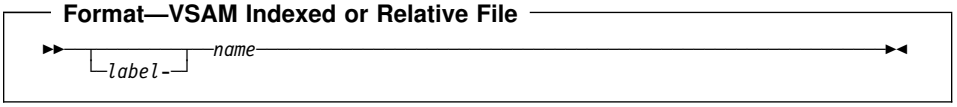
**Format—QSAM File**




**Format—VSAM Sequential File**




**Format—VSAM Indexed or Relative File**



## ASSIGN Clause

**label-**
Documents the device and device class to which a file is assigned. If specified, it must end with a hyphen.

**S-** For QSAM files, the S- (organization) field can be omitted.

**AS-**
For VSAM sequential files, the AS- (organization) field must be specified.

For VSAM indexed and relative files, the organization field must be omitted.

**name**
A required field that specifies the external name for this file. Under MVS it must be the name specified in the DD statement for this file.

The *name* must conform to the following rules of formation:

- If assignment-name-1 is a user-defined word:
  - The name can contain from 1 - 8 characters.
  - The name can contain the characters A-Z, a-z, 0-9.
  - The leading character must be alphabetic.
- If assignment-name-1 is a literal:
  - The name can contain from 1 - 8 characters.
  - The name can contain the characters A-Z, a-z, 0-9, @, #, $.
  - The leading character must be alphabetic.

For both user-defined words and literals, the compiler folds *name* to upper case to form the DD name for the file.

In a sort or merge file, *name* is treated as a comment.

If the file connector referenced by file-name-1 in the SELECT clause is an external file connector, all file control entries in the run unit that reference this file connector must have a consistent specification for assignment-name-1 in the ASSIGN clause. For QSAM files and VSAM indexed and relative files, the name specified on the first assignment-name-1 must be identical. For VSAM sequential files, it must be specified as AS-name.

## AIX, OS/2, and Windows Syntax

**assignment-name-1**
Can be either a user-defined word or a literal.

**User-defined word**
Assignment-name-1 must follow the rules for a COBOL word. The *name* component of the assignment name can be up to 30 characters in length. A user-defined word is treated as one of the following:

- **Environment variable name**— At program initialization, the *name* is used as an environment variable. If the environment variable value is set, that value is treated as the system file name optionally preceded by the file system ID. See “Assignment Name

for Data-Names and Environment Variables" on page 106 for details.

- **System file ID of the platform**— If the environment variable indicated by the *name* is not set, the user-defined word is treated as the system file name, optionally preceded by the file system ID and a comment character string. See "Assignment Name for Non-Environment Variables and Literals" for details.

**Literal**

Assignment-name-1 is treated as the actual file ID for the platform. Assignment-name-1 must follow the rules for a COBOL literal with the length of one to 160 characters. See "Assignment Name for Non-Environment Variables and Literals" for details.

All characters specified within the literal delimiters are used without any mapping.

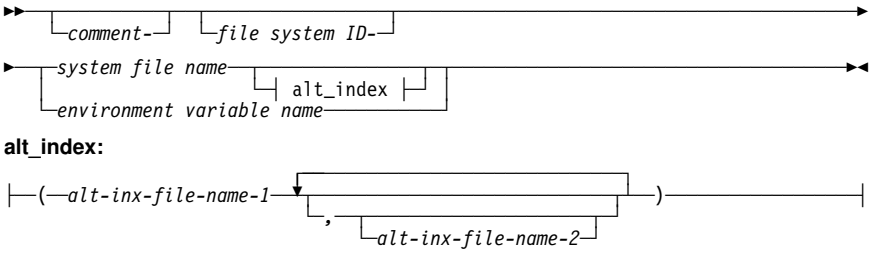
x **USING data-name-9**

x Must be defined as a data-name in the Data Division. The content of the data-
x name is evaluated when OPENed to identify the assignment name. See "Assign-
x ment Name for Data-Names and Environment Variables" on page 106 for details.

## Assignment Name for Non-Environment Variables and Literals

If a literal or non-data-name word is specified for the *name*, the assignment name is processed as follows:

**ASSIGNment name format**




**Comment**

All characters to the left of the system file ID are treated as comments. Comments can be hyphenated, for example, `my-comment` or `this-is-my-comment`.

**File system ID**

The first three characters of the file system ID- are used to determine the file system identifier. If the character string for the file system ID is less than three characters, then entire character string (along with any character strings to the left of it) is treated as a comment. If you include comments (hyphenated or not), you must include the separating hyphen between the comment and the file system ID.

For example, take the following two assignment-name formats:

```
my-comment-vsam-myfile
```

In this example, `my-comment` is the comment, `vsam` is the file system ID, and `myfile` is the system file or environment variable name.

```
my-comment-am-myfile
```

In this example, `my-comment-am` is the comment, and `myfile` is the system file or environment variable name.

**System file name / Environment variable name**

If the assignment name is not specified in the literal form and the environment variable matching the character string is found at run time, the environment variable value is used to identify the file system and the system file name. Otherwise, the character string is used as the system file name.

**Specifying alternate indexes**mdash. The compiler normally assigns default alternate index file names; however, you must override the default assignment when:

- The file is not a local VSAM file and has different alternate index file name specification rules. For example, an SFS file where SFS requires an alternate index file name to start with the base file name followed by ; followed by a character string of your choice.
- The file already exists and has alternate index files with names not corresponding to the default alternate index file names that are assigned by the compiler. For example, a remote MVS VSAM file or a local VSAM file create through a different language, such as PL/I.

If specifying alternate index names, they must be specified in the same order as the alternate record keys are specified in the source program. You can omit alternate index names, but any other alternate index names must correspond to the position in the file definition. The following example shows how to specify the first and third alternate index names:

```
base-file-name(first-index-file-name,,third-index-file-name)
```

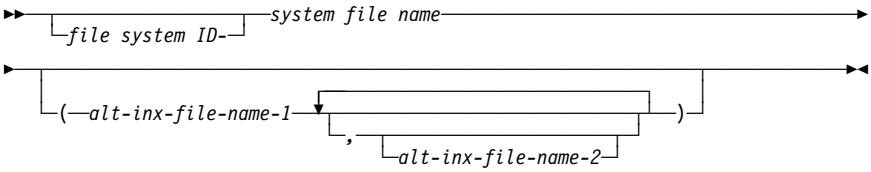
In the above example, the compiler will assign a default file name for the second alternate index file.

Alternate index file names are ignored for file systems that do not require separate alternate index files, such as the STL file system.

## Assignment Name for Data-Names and Environment Variables

If the environment variable or data-name is specified for the assignment name, the data-name value or the environment variable value is processed as follows:

**Environment variable and data name value format**




**file system ID** If the file system ID is specified explicitly using the environment variable value or the data-name value, that specification for the file system overrides any file system specification made by the ASSIGNment name.

The environment variable value for a file is obtained when the program containing the file is first invoked (or called) in its initial state. This value is kept for the file for subsequent calls to the program in the last used state.

The value of the file ID specified with a data-name is obtained when the file is OPENed. On each subsequent OPEN for the file, the value is reobtained.

File declarations for an external file must have the same file system identifier. If they are not, the error is caught during run time, and the application is terminated with an error message.

**system file name** If there is a hyphen in the environment variable or the data name value, the first three characters to the left of the left-most hyphen are treated as the file system identifier. The character string to right of the left most hyphen is then used as the system file name (possibly including drive and path names).

If there is no hyphen or the character string to the left of the left-most hyphen is less than three characters long, the entire character string is used as the system file name (possibly including drive and path names).

For information on specifying alternate indexes, see page 106.

## RESERVE Clause

x WRKSTN Under AIX, OS/2, and Windows, the RESERVE clause is not supported for
x line sequential files. It is treated as a comment for sequential, relative and indexed files.

The RESERVE clause allows the user to specify the number of input/output buffers to be allocated at run-time for the files.

If the RESERVE clause is omitted, the number of buffers at run time is taken from the DD statement when running under MVS. If none is specified, the system default is taken.

## ORGANIZATION Clause

If the file connector referenced by file-name-1 in the SELECT clause is an external file connector, all file control entries in the run unit that reference this file connector must have the same value for the integer specified in the RESERVE clause.

---

## ORGANIZATION Clause

The ORGANIZATION clause identifies the logical structure of the file. The logical structure is established at the time the file is created and cannot subsequently be changed.

You can find a discussion of the different ways in which data can be organized and of the different access methods that you can use to retrieve the data under "File Organization and Access Modes" on page 113.

**ORGANIZATION IS SEQUENTIAL (Format 1)**
A predecessor-successor relationship among the records in the file is established by the order in which records are placed in the file when it is created or extended.

**ORGANIZATION IS INDEXED (Format 2)**
The position of each logical record in the file is determined by indexes created with the file and maintained by the system. The indexes are based on embedded keys within the file's records.

**ORGANIZATION IS RELATIVE (Format 3)**
The position of each logical record in the file is determined by its relative record number.

x **ORGANIZATION IS LINE SEQUENTIAL (Format 4) (Workstation Only)**
x Under AIX, OS/2, and Windows, a predecessor-successor relationship among the
x records in the file is established by the order in which records are placed in the file
| when it is created or extended. A record in a LINE SEQUENTIAL file can consist
| only of printable characters.

If you omit the ORGANIZATION clause, the compiler assumes ORGANIZATION IS SEQUENTIAL.

If the file connector referenced by file-name-1 in the SELECT clause is an external file connector, all file control entries in the run unit that reference this file connector must have the same organization.

## File Organization

You establish the organization of the data when you create the file. Once the file has been created, you can expand the file, but you cannot change the organization.

### Sequential Organization

The physical order in which the records are placed in the file determines the sequence of records. The relationships among records in the file do not change, except that the file can be extended. Records can be fixed-length or variable-length; there are no keys.

Each record in the file, except the first, has a unique predecessor record, and each record, except the last, also has a unique successor record.

### Indexed Organization

Each record in the file has one or more embedded keys (referred to as key data items); each key is associated with an index. An index provides a logical path to the data records, according to the contents of the associated embedded record key data items. Indexed files must be direct-access storage files. Records can be fixed-length or variable-length.

Each record in an indexed file must have an embedded prime key data item. When records are inserted, updated, or deleted, they are identified solely by the values of their prime keys. Thus, the value in each prime key data item must be unique and must not be changed when the record is updated. You tell COBOL the name of the prime key data item on the RECORD KEY clause of the FILE-CONTROL paragraph.

In addition, each record in an indexed file can contain one or more embedded alternate key data items. Each alternate key provides another means of identifying which record to retrieve. You tell COBOL the name of any alternate key data items on the ALTERNATE RECORD KEY clause of the FILE-CONTROL paragraph.

The key used for any specific input-output request is known as the **key of reference**.

### Relative Organization

Think of the file as a string of record areas, each of which contains a single record. Each record area is identified by a relative record number; the access method stores and retrieves a record, based on its relative record number. For example, the first record area is addressed by relative record number 1, and the 10th is addressed by relative record number 10. The physical sequence in which the records were placed in the file has no bearing on the record area in which they are stored, and thus on each record's relative record number. Relative files must be direct-access files. Records can be fixed-length or variable-length.

x ### Line Sequential Organization (Workstation Only)

x In a line sequential file, each record contains a sequence of characters ending with a
x record terminator. The terminator is not counted in the length of the record. When
x records are written to the file, trailing blanks are removed.

x When reading the record, characters are read one at a time into the record area until:

- x The first record terminator is encountered. The record terminator is discarded and
  x the remainder of the record is filled with spaces.
- x The entire record area is filled with characters. If the first unread character is the
  x record terminator, it is discarded. Otherwise, the first unread character becomes
  x the first character read by the next READ statement.

x Records written to line sequential files must consist of USAGE...DISPLAY and/or
x DISPLAY-1 data items. If external decimal data is defined with a non-separate sign,

x the sign must be in the preferred sign representation (for example, X'39' for +9 or X'79'
x for -9).

x For line sequential files, the native byte stream file support of the platform is used. Line
x sequential files should contain only printable characters and the record terminator.

x The following are not supported for line sequential files:

x • APPLY WRITE ONLY clause
x • BLOCK CONTAINS clause
x • CODE-SET clause
x • DATA RECORDS clause
x • FILE STATUS value 39 (fixed file attribute conflict)
x • LABEL RECORDS clause
x • LINAGE clause
x • OPEN I-O option
x • PADDING CHARACTER clause
x • RECORD CONTAINS 0 clause
x • RECORD CONTAINS clause (format 3)
x • RECORD DELIMITER clause
x • RECORDING MODE clause
x • RERUN clause
x • RESERVE clause
x • REVERSED phrase of OPEN statement
x • REWRITE statement
x • VALUE OF clause of file description entry
x • WRITE...AT END-OF-PAGE
x • WRITE...BEFORE ADVANCING

x For more details on line sequential files, see “Line Sequential Organization (Workstation
x Only)” on page 109.

Under AIX, OS/2, and Windows for other files (sequential, relative, and indexed), the following language elements are treated as comments:

x • APPLY WRITE ONLY clause
• BLOCK CONTAINS clause
• CLOSE....FOR REMOVAL
• CLOSE....WITH NO REWIND
• CODE-SET clause
• DATA RECORDS clause
• LABEL RECORDS clause
• MULTIPLE FILE TAPE clause
• OPEN...REVERSE
• PADDING CHARACTER clause
x • PASSWORD clause
• RECORD CONTAINS 0 clause
• RECORD DELIMITER clause
x • RECORDING MODE clause (for relative and indexed files)
• RERUN clause

- RESERVE clause
- SAME AREA clause
- SAME SORT AREA clause
- SAME SORT-MERGE AREA clause
- VALUE OF clause of file description entry

No error messages are generated (with the exception of the *data name* option for the LABEL RECORDS, USE...AFTER...LABEL PROCEDURE, and GO TO MORE-LABELS clauses.

## PADDING CHARACTER Clause

x WRKSTN Under AIX, OS/2, and Windows, the PADDING CHARACTER clause is not
x supported for line sequential files. It is treated as a comment for sequential, relative and indexed files.

The PADDING CHARACTER clause specifies the character which is to be used for block padding on sequential files.

**data-name-5**
Must be defined in the Data Division as an alphanumeric one-character data item, and must not be defined in the File Section. Data-name-5 can be qualified.

**literal-2**
Must be a 1-character nonnumeric literal.

For EXTERNAL files, if data-name-5 is specified, it must reference an external data item.

The PADDING CHARACTER clause is syntax checked, but no compile-time or run-time verification checking is done, and the clause has no effect on the execution of the program.

## RECORD DELIMITER Clause

x WRKSTN Under AIX, OS/2, and Windows the RECORD DELIMITER clause is not sup-
x ported for line sequential files. It is treated as a comment for sequential, relative and indexed files.

The RECORD DELIMITER clause indicates the method of determining the length of a variable-length record on an external medium. It can be specified only for variable-length records.

**STANDARD-1**
If STANDARD-1 is specified, the external medium must be a magnetic tape file.

**assignment-name-2**
Can be any COBOL word.

The RECORD DELIMITER clause is syntax checked, but no compile-time or run-time

verification checking is done, and the clause has no effect on the execution of the program.

## ACCESS MODE Clause

The ACCESS MODE clause defines the manner in which the records of the file are made available for processing. If the ACCESS MODE clause is not specified, sequential access is assumed.

x For sequentially accessed relative files, the ACCESS MODE clause does not have to
x precede the RELATIVE KEY clause.

**ACCESS MODE IS SEQUENTIAL**
Can be specified in all four formats.

**Format 1—Sequential**
Records in the file are accessed in the sequence established when the file is created or extended. Format 1 supports only sequential access.

**Format 2—Indexed**
Records in the file are accessed in the sequence of ascending record key values according to the collating sequence of the file.

**Format 3—Relative**
Records in the file are accessed in the ascending sequence of relative record numbers of existing records in the file.

x **Format 4—Line Sequential (Workstation Only)**
x WRKSTN Records in the file are accessed in the sequence established when
x the file is created or extended. Format 4 supports only sequential access.

**ACCESS MODE IS RANDOM**
Can be specified in Formats 2 and 3 only.

**Format 2—Indexed**
The value placed in a record key data item specifies the record to be accessed.

**Format 3—Relative**
The value placed in a relative key data item specifies the record to be accessed.

**ACCESS MODE IS DYNAMIC**
Can be specified in Formats 2 and 3 only.

**Format 2—Indexed**
Records in the file can be accessed sequentially or randomly, depending on the form of the specific input-output statement used.

**Format 3—Relative**
Records in the file can be accessed sequentially or randomly, depending on the form of the specific input-output request.

## File Organization and Access Modes

**File organization** is the permanent logical structure of the file. You tell the computer how to retrieve records from the file by specifying the **access mode** (sequential, random, or dynamic). For details on the access methods and data organization, see Table 9 on page 99.

**Note:** Sequentially organized data can only be accessed sequentially; however, data that has indexed or relative organization can be accessed with any of the three access methods.

## Access Modes

**Sequential-Access Mode**
Allows reading and writing records of a file in a serial manner; the order of reference is implicitly determined by the position of a record in the file.

**Random-Access Mode**
Allows reading and writing records in a programmer-specified manner; the control of successive references to the file is expressed by specifically defined keys supplied by the user.

**Dynamic-Access Mode**
Allows the specific input-output statement to determine the access mode. Therefore, records can be processed sequentially and/or randomly.

For EXTERNAL files, every file control entry in the run unit that is associated with that external file must specify the same access mode. In addition, for relative file entries, data-name-4 must reference an external data item and the RELATIVE KEY phrase in each associated file control entry must reference that same external data item in each case.

## Relationship Between Data Organizations and Access Modes

The following lists which access modes are valid for each type of data organization.

x **Sequential and Line Sequential Files**
Files with sequential organization can be accessed only sequentially. The sequence in which records are accessed is the order in which the records were originally written.

**Indexed Files**
All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is the ascending order (or optionally under AIX, OS/2, and Windows, descending order) of the record key value. The order of retrieval within a set of records having duplicate alternate record key values is the order in which records were written into the set.

In the random access mode, you control the sequence in which records are accessed. The desired record is accessed by placing the value of its key(s) in the RECORD KEY data item (and the ALTERNATE RECORD KEY data item). If a set

## RECORD KEY Clause

of records has duplicate alternate record key values, only the first record written is available.

In the dynamic access mode, you can change, as necessary, from sequential access to random access, using appropriate forms of input-output statements.

**Relative Files**

All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is the ascending order (or optionally under AIX, OS/2, and Windows, descending order) of the relative record numbers of all records that currently exist within the file.

In the random access mode, you control the sequence in which records are accessed. The desired record is accessed by placing its relative record number in the RELATIVE KEY data item; the RELATIVE KEY must not be defined within the record description entry for this file.

In the dynamic access mode, you can change, as necessary, from sequential access to random access, using the appropriate forms of input-output statements.

---

## RECORD KEY Clause

The RECORD KEY clause (Format 2) specifies the data item within the record that is the prime RECORD KEY for an indexed file. The values contained in the prime RECORD KEY data item must be unique among records in the file.

**data-name-2**

The prime RECORD KEY data item. It must be described as an alphanumeric item within a record description entry associated with the file.

x As an IBM extension, data-name-2 can be numeric, numeric-edited, alphanumeric-
x edited, alphabetic, floating-point (both external and internal), or a DBCS data item.
x The key is treated as an alphanumeric item for the input and output statements for
x the file named in the SELECT clause. When you specify data-name-2 as a DBCS
x data item, a key specified on the READ statement must also be a DBCS data item.

Data-name-2 must not reference a group item that contains a variable occurrence data item. Data-name-2 can be qualified.

x As an IBM extension, if the indexed file contains variable-length records,
x data-name-2 need not be contained within the first “x” character positions of the
x record, where “x” equals the minimum record size specified for the file. That is,
x data-name-2 can be beyond the first “x” character positions of the record, but this
x is not recommended.

The data description of data-name-2 and its relative location within the record must be the same as those used when the file was defined.

If the file has more than one record description entry, data-name-2 need only be described in one of these record description entries. The identical character positions referenced by data-name-2 in any one record description entry are implicitly referenced as keys for all other record description entries of that file.

For EXTERNAL files, all file description entries in the run unit that are associated with the EXTERNAL file must specify the same data description entry for data-name-2 with the same relative location within the associated record.

x The requirement for identical data description entries is not enforced, but the key must
x have the same relative location in the records, as well as the same length.

## ALTERNATE RECORD KEY Clause

The ALTERNATE RECORD KEY clause (Format 2) specifies a data item within the record that provides an alternative path to the data in an indexed file.

**data-name-3**
An ALTERNATE RECORD KEY data item. It must be described as an alphanumeric item within a record description entry associated with the file. It must not reference a group item that contains a variable occurrence data item. Data-name-3 can be qualified.

If the indexed file contains variable-length records, data-name-3 must be contained within the first "x" character positions of the record, where "x" equals the minimum record size specified for the file.

x Data-name-3 can be a numeric, numeric-edited, alphanumeric-edited, alphabetic,
x floating-point (both external and internal), or DBCS data item. The key is treated
x as an alphanumeric item for the input and output statements for the file named in
x the SELECT clause.

If the file has more than one record description entry, data-name-3 need be described in only one of these record description entries. The identical character positions referenced by data-name-3 in any one record description entry are implicitly referenced as keys for all other record description entries of that file.

The data description of data-name-3 and its relative location within the record must be the same as those used when the file was defined. The number of alternate record keys for the file must also be the same as that used when the file was created.

The leftmost character position of data-name-3 must not be the same as the leftmost character position of the RECORD KEY or of any other ALTERNATE RECORD KEY.

If the DUPLICATES phrase is not specified, the values contained in the ALTERNATE RECORD KEY data item must be unique among records in the file.

If the DUPLICATES phrase is specified, the values contained in the ALTERNATE RECORD KEY data item can be duplicated within any records in the file. In sequential access, the records with duplicate keys are retrieved in the order in which they were placed in the file. In random access, only the first record written of a series of records with duplicate keys can be retrieved.

For EXTERNAL files, all file description entries in the run unit that are associated with the EXTERNAL file must specify the same data description entry for data-name-3, the

## PASSWORD Clause

same relative location within the associated record, the same number of alternate record keys, and the same DUPLICATES phrase.

x The requirement for identical data description entries is not enforced, but the key must
x have the same relative location in the records, as well as the same length.

## RELATIVE KEY Clause

The RELATIVE KEY clause (Format 3) identifies a data-name that specifies the relative record number for a specific logical record within a relative file.

**data-name-4**
Must be defined as an unsigned integer data item whose description does not contain the PICTURE symbol P. Data-name-4 must not be defined in a record description entry associated with this relative file. That is, the RELATIVE KEY is **not** part of the record. Data-name-4 can be qualified.

Data-name-4 is required for ACCESS IS SEQUENTIAL only when the START statement is to be used. It is always required for ACCESS IS RANDOM and ACCESS IS DYNAMIC. When the START statement is issued, the system uses the contents of the RELATIVE KEY data item to determine the record at which sequential processing is to begin.

If a value is placed in data-name-4, and a START statement is not issued, the value is ignored and processing begins with the first record in the file.

If a relative file is to be referenced by a START statement, you must specify the RELATIVE KEY clause for that file.

For EXTERNAL files, data-name-4 must reference an external data item and the RELATIVE KEY phrase in each associated file control entry must reference that same external data item in each case.

The ACCESS MODE IS RANDOM clause must not be specified for file-names specified in the USING or GIVING phrase of a SORT or MERGE statement.

## x PASSWORD Clause

x WRKSTN Under AIX, OS/2, and Windows the PASSWORD clause is treated as a
x comment.

x The PASSWORD clause controls access to files.

x **data-name-6**
x **data-name-7**
x Password data items. Each must be defined in the Working-Storage Section (of the
x Data Division) as an alphanumeric item. The first 8 characters are used as the
x password; a shorter field is padded with blanks to 8 characters. Each password
x data item must be equivalent to one that is externally defined.

x When the PASSWORD clause is specified, at object time the PASSWORD data item
x must contain the valid password for this file before the file can be successfully opened.

**Format 1 Considerations**:

The PASSWORD clause is not valid for QSAM sequential files.

**Format 2 and 3 Considerations**:

When the PASSWORD clause is specified, it must immediately follow the RECORD KEY or ALTERNATE RECORD KEY data-name with which it is associated.

For indexed files, if the file has been completely predefined to VSAM, only the PASSWORD data item for the RECORD KEY need contain the valid password before the file can be successfully opened at file creation time.

For any other type of file processing (including the processing of dynamic CALLs at file creation time through a COBOL object-time subroutine), every PASSWORD data item for this file must contain a valid password before the file can be successfully opened, whether or not all paths to the data are used in this object program.

For EXTERNAL files, data-name-6 and data-name-7 must reference external data items. The PASSWORD clauses in each associated file control entry must reference the same external data items.

## LOCK MODE Clause

MVS/VM The LOCK MODE clause is not applicable for MVS and VM.

The LOCK MODE clause specifies whether a file is in exclusive or shareable mode. A file in exclusive mode is open to one file connector only. A file in shareable mode is available to any number of file connectors that do not require exclusive mode.

A file is in exclusive mode if the LOCK MODE clause is omitted (as long as the file is not opened for input).

A file is in shareable mode when it is opened for input or when the LOCK MODE IS AUTOMATIC clause is specified.

Do not specify the LOCK MODE IS AUTOMATIC clause if the file is specified in a USING or GIVING phrase of a SORT or MERGE statement.

The WITH LOCK ON RECORD clause is for documentation purposes only.

## Other Statements Affecting Record Locking

Table 10 lists the statements that can affect record locking.

*Table 10. Statements Affecting Record Locking*

| Statement | Comments |
|---|---|
| CLOSE | After you successfully CLOSE a file, any record and file locks held by the file connector on the closed file are released. |
| DELETE | You cannot DELETE a record that any other file connector has LOCKed. |
| OPEN | If you attempt to OPEN a file that another file connector has LOCKed, the OPEN fails and you receive a 'file locked' file status (98). |
| READ | For files opened for INPUT, READ statements will not acquire a record lock.<br><br>If you attempt to READ a record that another file connector has LOCKed, the READ fails and you receive a 'record locked' file status (FS 99). For a sequential READ, the setting of the file position indicator is unaffected. For a random READ, the setting of the file position indicator is unspecified.<br><br>When you specify the READ statement at the end of the file (when no more records exist), the AT END condition is returned regardless of any sharing of the file. This situation can occur if the file is opened in EXTEND mode by another file connector.<br><br>If you OPEN the file for I-O and specify the LOCK MODE IS AUTOMATIC clause, each record is locked as it is read and released by the next I-O statement accessing the file connector. |
| REWRITE | You cannot specify the REWRITE statement for a record that another file connector has LOCKed (the file is exclusive).<br><br>If LOCK MODE IS AUTOMATIC is specified (the file is shareable), you can use the REWRITE statement to release a record that is LOCKed. |
| START | You cannot use the START statement to LOCK a record or to detect if a record is LOCKed. However, the START statement will release an existing LOCKed record if you have specified the LOCK MODE IS AUTOMATIC clause. |
| WRITE | If two or more file connectors add records to a file by sharing the file after opening it in EXTEND mode, the following occurs:<br>• **Sequential files and line sequential files:**— the records are in an unspecified order.<br>• **Relative files:**—the relative key values returned are ascending but not necessarily consecutive.<br>• **Indexed files:**—the order of the alternate keys allowing for duplicates is unspecified.<br><br>When you specify LOCK MODE IS AUTOMATIC, a successful WRITE statement releases a LOCKed record. |

## FILE STATUS Clause

The FILE STATUS clause monitors the execution of each input-output operation for the file.

When the FILE STATUS clause is specified, the system moves a value into the status key data item after each input-output operation that explicitly or implicitly refers to this file. The value indicates the status of execution of the statement. (See the "Status Key" description under "Common Processing Facilities" on page 254.)

**data-name-1**

The status key data item can be defined in the Working-Storage, Local-Storage, or Linkage sections as either of the following:

- A 2-character alphanumeric item
- A 2-character numeric data item, with explicit or implicit USAGE IS DISPLAY. It is treated as an alphanumeric item.

  **Note:** Data-name-1 must not contain the PICTURE symbol `'P'`.

Data-name-1 can be qualified.

The status key data item must not be variably located; that is, the data item cannot follow a data item containing an OCCURS DEPENDING ON clause.

**data-name-8**

Represents information returned from the file system. Since the definitions are specific to the file systems and platforms, applications that depend on the specific values in data-name-8 might not be portable across platforms.

MVS/VM Under MVS and VM, data-name-8 must be defined as a group item of 6 bytes in the Working-Storage or Linkage Section of the Data Division.

Specify data-name-8 only if the file is a VSAM file (that is, ESDS, KSDS, RRDS).

On MVS and VM, for VSAM Files the 6 byte VSAM return code is comprised of the following:

- The first 2 bytes of data-name-8 contain the VSAM **return code** in binary notation. The value for this code is defined (by VSAM) as 0, 8, or 12.
- The next 2 bytes of data-name-8 contain the VSAM **function code** in binary notation. The value for this code is defined (by VSAM) as 0, 1, 2, 3, 4, or 5.
- The last 2 bytes of data-name-8 contain the VSAM **feedback code** in binary notation. The code value is 0 through 255.

If VSAM returns a nonzero return code, data-name-8 is set.

If FILE STATUS is returned without having called VSAM, data-name-8 is zero.

If data-name-1 is set to zero, the content of data-name-8 is undefined. VSAM status return code information is available without transformation in the currently defined COBOL FILE STATUS code. User identification and handling of exception conditions are allowed at the same level as that defined by VSAM.

## I-O-CONTROL Paragraph

**Function code** and **feedback code** are set if and only if the **return code** is set to nonzero. If they are referenced when the return code is set to zero, the contents of the fields are not dependable.

Definitions of values in the **return code**, **function code**, and **feedback code** fields are defined by VSAM. There are no COBOL additions, deletions, or modifications to the VSAM definitions. For more information, see *VSAM Administration: Macro Instruction Reference*.

WRKSTN Under AIX, OS/2, and Windows, how you define data-name-8 is dependent on the file system you are using.

**Btrieve, STL, and Native Platform File Systems**

You must define data-name-8 with PICTURE 9(6) and USAGE DISPLAY attributes. However, you can define an additional field with PICTURE X(n). The file system defines the feedback values, which are converted to the six digit external decimal representation with leading zeros, when the file systems feedback value is less than 100000. If you have defined an additional field using PICTURE X(n), then X(n) contains additional information describing any non-zero feedback code. (For most programs, an 'n' value of 100 should be adequate to show the complete message text. If the file is defined with a large number of alternate keys then allow 100 bytes plus 20 bytes per alternate key.)

**VSAM File Systems**

You must define data-name-8 with PICTURE X(n) and USAGE DISPLAY attributes, where 'n' is 6 or greater. The PICTURE string value represents the first 'n' bytes of the VSAM reply message structure (defined by VSAM). If the size of the reply message structure (m) is shorter than 'n', only the first 'm' bytes contain useful information.

For information on VSAM file handling on AIX and OS/2, see:

- For AIX: *SMARTdata UTILITIES for AIX: VSAM in a Distributed Environment*
- For OS/2: *SMARTdata UTILITIES for OS/2: VSAM in a Distributed Environment*

## I-O-CONTROL Paragraph

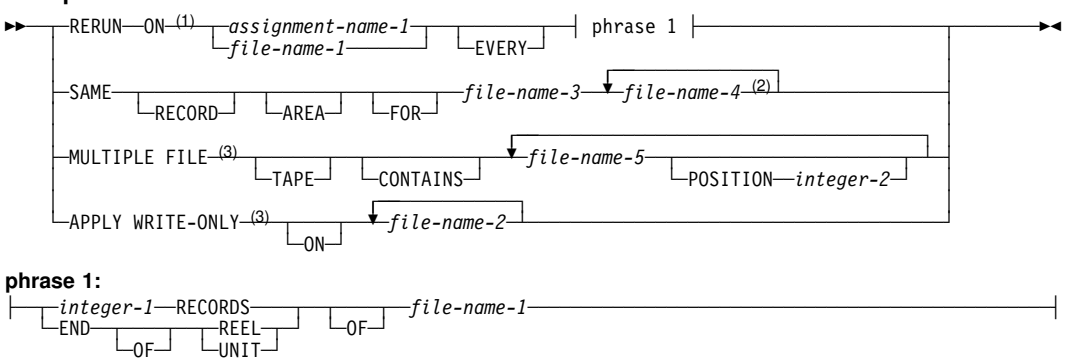
The I-O-CONTROL paragraph of the Input-Output Section specifies when checkpoints are to be taken and the storage areas to be shared by different files. This paragraph is optional in a COBOL program.

The key word I-O-CONTROL can appear only once, at the beginning of the paragraph. The word I-O-CONTROL must begin in Area A, and must be followed by a separator period.

Each clause within the paragraph can be separated from the next by a separator comma or a separator semicolon. The order in which I-O-CONTROL paragraph

clauses are written is not significant. The I-O-CONTROL paragraph ends with a separator period.

**Sequential I-O-Control Entries**




**phrase 1:**



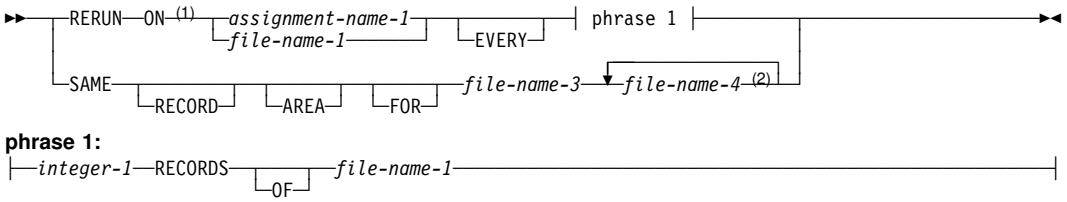

**Notes:**

1 ON is optional as an IBM extension.

2 File-name-4 is optional as an IBM extension.

3 The MULTIPLE FILE clause and APPLY WRITE-ONLY clause are not supported for MVS VSAM files and are treated as comments on AIX, OS/2, and Windows.

**Relative and Indexed I-O-Control Entries**




**phrase 1:**

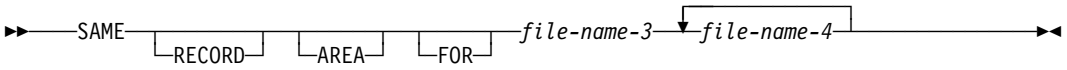



**Notes:**

1 ON is optional as an IBM extension.

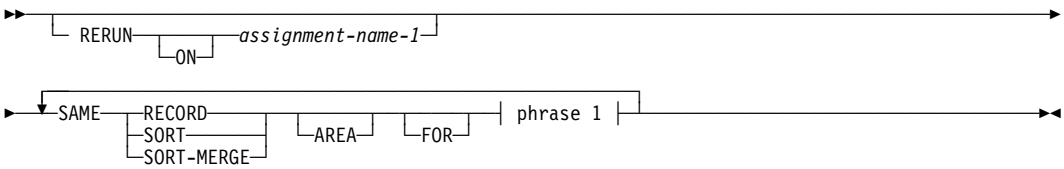
2 File-name-4 is optional as an IBM extension.

**Line Sequential I-O-Control Entries (Workstation Only)**

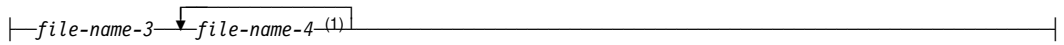


## RERUN Clause

**Sort Merge I-O-Control Entries (MVS and VM Only)**




**phrase 1:**




**Note:**

1 File-name-4 is optional as an IBM extension.

## RERUN Clause

WRKSTN Under AIX, OS/2, and Windows, the RERUN clause is not supported for line sequential files or for programs compiled with the THREAD compiler option. If you use NOTHREAD, the RERUN clause is treated as a comment.

The RERUN clause specifies that checkpoint records are to be taken. Subject to the restrictions given with each phrase, more than one RERUN clause can be specified.

For information regarding the checkpoint data set definition and the checkpoint method required for complete compliance to the COBOL 85 Standard, see *IBM COBOL for MVS & VM Programming Guide*.

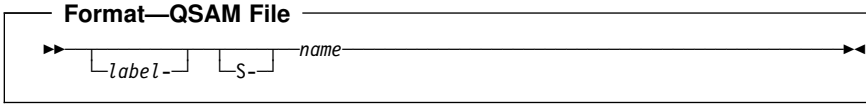
Do not use the RERUN clause:

- On files with the EXTERNAL attribute
- In programs with the RECURSIVE attribute
- In programs compiled with the THREAD option (Workstation only)
- In methods

**file-name-1**
Must be a sequentially organized file.

**assignment-name-1**
The external data set for the checkpoint file. It must not be the same assignment-name as that specified in any ASSIGN clause throughout the entire program, including contained and containing programs. For QSAM files, it has the format:

**Format—QSAM File**




That is, it must be a QSAM file. It must reside on a tape or direct access device. See also Appendix E, “ASCII Considerations for MVS and VM” on page 539.

**VSAM and QSAM Considerations**:

The file named in the RERUN clause must be a file defined in the same program as the I-O-CONTROL paragraph, even if the file is defined as GLOBAL.

x **SORT/MERGE Considerations**:

x When the RERUN clause is specified in the I-O-CONTROL paragraph, checkpoint
x records are written at logical intervals determined by the sort/merge program during
x execution of each SORT or MERGE statement in the program. When it is omitted,
x checkpoint records are not written.

x There can be only **one** SORT/MERGE I-O-CONTROL paragraph in a program, and
x it cannot be specified in contained programs. It will have a global effect on all
x SORT and MERGE statements in the program unit.

**EVERY integer-1 RECORDS**
A checkpoint record is to be written for every integer-1 record in file-name-1 that is processed.

When multiple integer-1 RECORDS phrases are specified, no two of them can specify the same file-name-1.

If you specify the integer-1 RECORDS phrase, you must specify assignment-name-1.

**EVERY END OF REEL/UNIT**
A checkpoint record is to be written whenever end-of-volume for file-name-1 occurs. The terms REEL and UNIT are interchangeable.

When multiple END OF REEL/UNIT phrases are specified, no two of them can specify the same file-name-1.

The END OF REEL/UNIT phrase can only be used if file-name-1 is a sequentially organized file.

## SAME AREA Clause

WRKSTN Under AIX, OS/2, and Windows, the SAME AREA clause is treated as a comment.

The SAME AREA clause specifies that two or more files, that do not represent sort or merge files, are to use the same main storage area during processing.

The files named in a SAME AREA clause need not have the same organization or access.

**file-name-3**
**file-name-4**
Must be specified in the FILE-CONTROL paragraph of the same program. File-name-3 and file-name-4 cannot reference an external file connector.

- For QSAM files, the SAME clause is treated as documentation.

## SAME RECORD AREA Clause

- For MVS VSAM files, the SAME clause is treated as if equivalent to the SAME RECORD AREA.

More than one SAME AREA clause can be included in a program. However:

- A specific file-name must not appear in more than one SAME AREA clause.
- If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, the SAME RECORD AREA clause can contain additional file-names that do not appear in the SAME AREA clause.
- The rule that in the SAME AREA clause only one file can be open at one time takes precedence over the SAME RECORD AREA rule that all the files can be open at the same time.

---

## SAME RECORD AREA Clause

The SAME RECORD AREA clause specifies that two or more files are to use the same main storage area for processing the current logical record. All of the files can be open at the same time. A logical record in the shared storage area is considered to be both of the following:

- A logical record of each opened output file in the SAME RECORD AREA clause
- A logical record of the most recently read input file in the SAME RECORD AREA clause.

More than one SAME RECORD AREA clause can be included in a program. However:

- A specific file-name must not appear in more than one SAME RECORD AREA clause.
- If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, the SAME RECORD AREA clause can contain additional file-names that do not appear in the SAME AREA clause.
- The rule that in the SAME AREA clause only one file can be open at one time takes precedence over the SAME RECORD AREA rule that all the files can be open at the same time.
- If the SAME RECORD AREA clause is specified for several files, the record description entries or the file description entries for these files must not include the GLOBAL clause.
- The SAME RECORD AREA clause must not be specified when the RECORD CONTAINS 0 CHARACTERS clause is specified.

The files named in the SAME RECORD AREA clause need not have the same organization or access.

## SAME SORT AREA Clause

WRKSTN Under AIX, OS/2, and Windows, the SAME SORT AREA clause is treated as a comment.

The SAME SORT AREA clause is syntax checked but has no effect on the execution of the program.

**file-name-3**
**file-name-4**
Must be specified in the FILE-CONTROL paragraph of the same program. File-name-3 and file-name-4 cannot reference an external file connector.

When the SAME SORT AREA clause is specified, at least one file-name specified must name a sort file. Files that are not sort files can also be specified. The following rules apply:

- More than one SAME SORT AREA clause can be specified. However, a given sort file must not be named in more than one such clause.
- If a file that is not a sort file is named in both a SAME AREA clause and in one or more SAME SORT AREA clauses, all the files in the SAME AREA clause must also appear in that SAME SORT AREA clause.
- Files named in a SAME SORT AREA clause need not have the same organization or access.
- Files named in a SAME SORT AREA clause that are not sort files do not share storage with each other unless the user names them in a SAME AREA or SAME RECORD AREA clause.
- During the execution of a SORT or MERGE statement that refers to a sort or merge file named in this clause, any nonsort or nonmerge files associated with file-names named in this clause must not be in the open mode.

## SAME SORT-MERGE AREA Clause

WRKSTN Under AIX, OS/2, and Windows, the SAME SORT-MERGE AREA clause is treated as a comment.

The SAME SORT-MERGE AREA clause is equivalent to the SAME SORT AREA clause.

## APPLY WRITE-ONLY Clause

## MULTIPLE FILE TAPE Clause

WRKSTN Under AIX, OS/2, and Windows, all files are treated as a single volume file. Any multiple volume files specified are treated as comments.

The MULTIPLE FILE TAPE clause (Format 1) specifies that two or more files share the same physical reel of tape.

This clause is syntax checked, but it has no effect on the execution of the program. The function is performed by the system through the LABEL parameter of the DD statement.

## APPLY WRITE-ONLY Clause

x WRKSTN Under AIX, OS/2, and Windows, the APPLY WRITE-ONLY clause is not supx ported for line sequential files. It is treated as a comment for sequential, relative and x indexed files.

x The APPLY WRITE-ONLY clause optimizes buffer and device space allocation for files x that have standard sequential organization, have variable-length records, and are x blocked. If you specify this phrase, the buffer is truncated only when the space availx able in the buffer is smaller than the size of the next record. Otherwise, the buffer is x truncated when the space remaining in the buffer is smaller than the maximum record x size for the file.

x APPLY WRITE-ONLY is effective only for QSAM files.

x **file-name-2**
x Each file must have standard sequential organization.

x APPLY WRITE-ONLY clauses must agree among corresponding external file x description entries. For an alternate method of achieving the APPLY WRITE-ONLY x results, see the description of the AWO compiler option in the *IBM COBOL for MVS & x VM Programming Guide*.

# Part 5. Data Division


**Data Division Overview** . . . . . 128
File Section . . . . . 129
Working-Storage Section . . . . . 129
x Local-Storage Section . . . . . 131
Linkage Section . . . . . 131
Data Types . . . . . 132
Data Relationships . . . . . 133

**Data Division—File Description Entries** . . . . . 140
File Section . . . . . 144
EXTERNAL Clause . . . . . 144
GLOBAL Clause . . . . . 145
BLOCK CONTAINS Clause . . . . . 145
RECORD Clause . . . . . 147
LABEL RECORDS Clause . . . . . 150
VALUE OF Clause . . . . . 151
DATA RECORDS Clause . . . . . 151
LINAGE Clause . . . . . 151
x RECORDING MODE Clause . . . . . 154
CODE-SET Clause . . . . . 156

**Data Division—Data Description Entry** . . . . . 157
Format 1 . . . . . 157
Format 2 . . . . . 158
Format 3 . . . . . 158
Level-Numbers . . . . . 158
BLANK WHEN ZERO Clause . . . . . 160
EXTERNAL Clause . . . . . 160
GLOBAL Clause . . . . . 161
JUSTIFIED Clause . . . . . 162
OCCURS Clause . . . . . 163
PICTURE Clause . . . . . 169
REDEFINES Clause . . . . . 186
RENAMES Clause . . . . . 190
SIGN Clause . . . . . 192
SYNCHRONIZED Clause . . . . . 193
USAGE Clause . . . . . 200
VALUE Clause . . . . . 207


© Copyright IBM Corp. 1991, 1996

# Data Division Overview

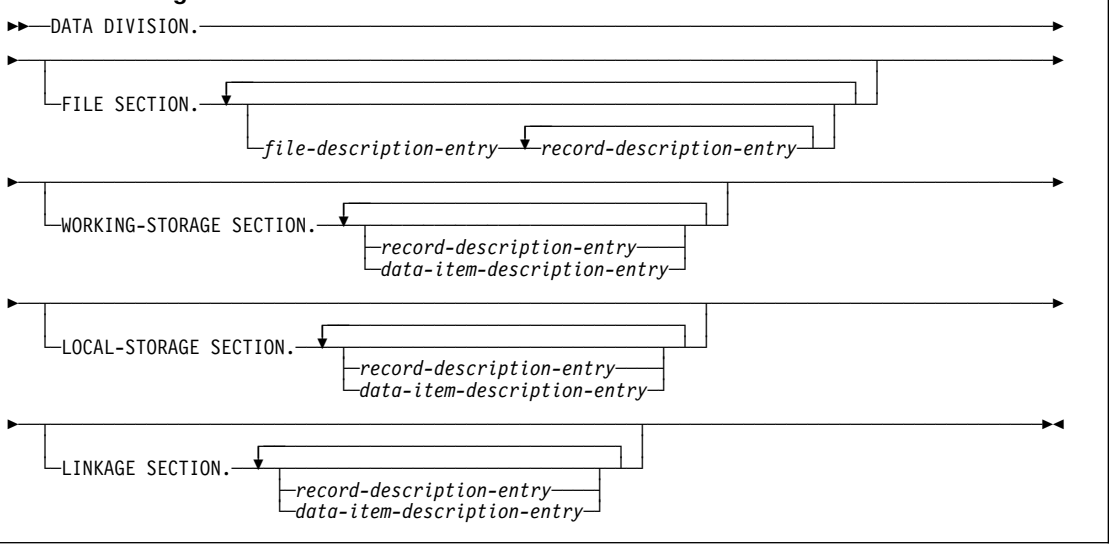
This section outlines the structure of the Data Division for programs, classes, and methods. Each section in the Data Division has a specific logical function within a COBOL source program or method and can be omitted when that logical function is not needed. If included, the sections must be written in the order shown. The Data Division is optional.

**Program Data Division**
The Data Division of a COBOL source program describes, in a structured manner, all the data to be processed by the object program.

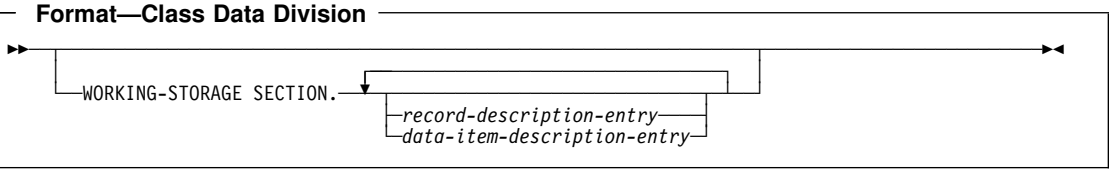
**Class Data Division**
The Class Data Division section contains data description entries for object-instance data. The Class Data Division contains only the Working-Storage Section.

**Method Data Division**
A method has two visible Data Divisions: the Class Data Division and the Method Data Division. If the same data-name is used in both the Class Data Division and the Method Data Division, when a method references the data-name, the data-name in the Method Data Division takes precedence.

**Format—Program and Method Data Division**




**Format—Class Data Division**



## File Section

The File Section defines the structure of data files. The File Section must begin with the header FILE SECTION, followed by a separator period.

**file-description-entry**
Represents the highest level of organization in the File Section. It provides information about the physical structure and identification of a file, and gives the record-name(s) associated with that file. For the format and the clauses required in a file description entry, see "Data Division—File Description Entries" on page 140.

**record-description-entry**
A set of data description entries (described in "Data Division—Data Description Entry" on page 157) that describe the particular record(s) contained within a particular file.

More than one record description entry can be specified; each is an alternative description of the same record storage area.

Data areas described in the File Section are not available for processing unless the file containing the data area is open.

x **Note:** A method File Section can define EXTERNAL files only. A single run-unit level
x file connector is shared by all programs and methods containing a declaration of a
x given EXTERNAL file.

## Working-Storage Section

The Working-Storage Section describes data records that are not part of data files but are developed and processed by a program or method. It also describes data items whose values are assigned in the source program or method and do not change during execution of the object program.

The Working-Storage Section must begin with the section header Working-Storage Section, followed by a separator period.

**Program Working-Storage**
The Working-Storage Section for programs (and methods) can also describe external data records, which are shared by programs and methods throughout the run-unit. All clauses that are used in record descriptions in the File Section as well as the VALUE and EXTERNAL clauses (which might not be specified in record description entries in the File Section) can be used in record descriptions in the Working-Storage Section.

x **Method Working-Storage**
x A single copy of the Working-Storage for a method is statically allocated and per-
x sists in a last-used state for the duration of the run-unit. The same single copy is
x used whenever the method is invoked, regardless of which object the method is
x invoked upon.

x If a VALUE clause is specified on a method Working-Storage data item, the data
x item is initialized to the VALUE clause value on the first invocation.

If the EXTERNAL attribute is specified on a data description entry in a method Working-Storage Section, a single copy of the storage for that data item is allocated once for the duration of the run-unit. That storage is shared by all programs and methods in the run-unit containing a definition for the external data item.

**Class Working-Storage**

A separate copy of the Class Working-Storage data items is allocated for each object instance and remains until that object is destroyed.

By default, Class Working-Storage data items are global to all of the methods introduced by the class.

To initialize instance data (Class Working-Storage data items), you can write a somInit method override. For an example of how to write an override method using somInit, see Figure 3. VALUE clauses are not supported for initializing instance data.

```
IDENTIFICATION DIVISION.
CLASS-ID.  OOClass INHERITS SOMObject.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS SOMObject IS "SOMObject"
    CLASS OOClass IS "OOClass".
DATA DIVISION.
Working-Storage Section.
01 instance-data  PIC X(3).
PROCEDURE DIVISION.

IDENTIFICATION DIVISION.
METHOD-ID. "somInit" OVERRIDE.
PROCEDURE DIVISION.
    MOVE "new" TO instance-data.
    EXIT METHOD.
END METHOD "somInit".

IDENTIFICATION DIVISION.
METHOD-ID. "MyMethod".
PROCEDURE DIVISION.
    IF instance-data = "new"
       CALL "Creating"
       MOVE "old" TO instance-data
    ELSE
       CALL "Existing"
    END-IF.
    EXIT METHOD.
END METHOD "MyMethod".

END CLASS OOClass.
```

*Figure 3. Example of a somInit Method Override*

The Working-Storage Section contains record description entries and data description entries for independent data items, called **data item description entries**.

**record-description-entry**
Data entries in the Working-Storage Section that bear a definite hierarchic relationship to one another must be grouped into records structured by level number. See "Data Division—Data Description Entry" on page 157 for description.

**data-item-description-entry**
Independent items in the Working-Storage Section that bear no hierarchic relationship to one another need not be grouped into records, provided that they do not need to be further subdivided. Instead, they are classified and defined as independent elementary items. Each is defined in a separate data-item description entry that begins with either the level number 77 or 01. See "Data Division—Data Description Entry" on page 157 for description.

x **Note:** The data description entries for a class differ from a program and method in
x that:

x • You cannot specify the EXTERNAL attribute in a data description entry.
x • The GLOBAL attribute has no effect.
x • You can only specify the VALUE clause on condition names.

## x Local-Storage Section

x The Local-Storage Section defines storage that is allocated and freed on a per-
x invocation basis. On each invocation, data items defined in the Local-Storage Section
x are reallocated and initialized to the value assigned in their VALUE clauses. Data
x items defined in the Local-Storage Section cannot specify the EXTERNAL clause.

x The Local-Storage Section must begin with the header LOCAL-STORAGE SECTION
x followed by a separator period.

x You can specify the Local-Storage Section in recursive programs, in non-recursive pro-
x grams, and in methods.

x **Note:** Method Local-Storage content is the same as a program Local-Storage content
x except that the GLOBAL attribute has no effect (since methods cannot be nested).

x A separate copy of the data defined in a method Local-Storage section is created each
x time the method is invoked. The storage allocated for the data is freed when the
x method returns.

## Linkage Section

The Linkage Section describes data made available from another program or method.

**record-description-entry**
See "Working-Storage Section" on page 129 for description.

## Data Types

**data-item-description-entry**
See "Working-Storage Section" on page 129 for description.

Record description entries and data item description entries in the Linkage Section provide names and descriptions, but storage within the program or method is not reserved because the data area exists elsewhere.

Any data description clause can be used to describe items in the Linkage Section with the following exceptions:

- You cannot specify the VALUE clause for items other than level-88 items.
- You cannot specify the EXTERNAL clause in the Linkage Section.

x As an IBM extension, you can specify the GLOBAL clause in the Linkage Section.
x (Note, the GLOBAL attribute has no effect for methods.)

---

## Data Types

Two types of data can be processed: file data and program data.

## File Data

File data is contained in files. (See "File Section" on page 144.) A **file** is a collection of data records existing on some input-output device. A file can be considered as a group of physical records; it can also be considered as a group of logical records. The Data Division describes the relationship between physical and logical records.

A **physical record** is a unit of data that is treated as an entity when moved into or out of storage. The size of a physical record is determined by the particular input-output device on which it is stored. The size does not necessarily have a direct relationship to the size or content of the logical information contained in the file.

A **logical record** is a unit of data whose subdivisions have a logical relationship. A logical record can itself be a physical record (that is, be contained completely within one physical unit of data); several logical records can be contained within one physical record, or one logical record can extend across several physical records.

**File description entries** specify the physical aspects of the data (such as the size relationship between physical and logical records, the size and name(s) of the logical record(s), labeling information, and so forth).

**Record description entries** describe the logical records in the file, including the category and format of data within each field of the logical record, different values the data might be assigned, and so forth.

After the relationship between physical and logical records has been established, only logical records are made available to you. For this reason, a reference in this manual to "records" means logical records, unless the term "physical records" is used.

## Program Data

Program data is created by a program, instead of being read from a file.

The concept of logical records applies to program data as well as to file data. Program data can thus be grouped into logical records, and be defined by a series of record description entries. Items that need not be so grouped can be defined in independent data description entries (called **data item description entries**).

## Data Relationships

The relationships among all data to be used in a program are defined in the Data Division, through a system of level indicators and level-numbers.

A **level indicator**, with its descriptive entry, identifies each file in a program. Level indicators represent the highest level of any data hierarchy with which they are associated; FD is the file description level indicator and SD is the sort-merge file description level indicator.

A **level-number**, with its descriptive entry, indicates the properties of specific data. Level-numbers can be used to describe a data hierarchy; they can indicate that this data has a special purpose, and while they can be associated with (and subordinate to) level indicators, they can also be used independently to describe internal data or data common to two or more programs. (See “Level-Numbers” on page 158 for level-number rules.)

## Levels of Data

After a record has been defined, it can be subdivided to provide more detailed data references.

For example, in a customer file for a department store, one complete record could contain all data pertaining to one customer. Subdivisions within that record could be: customer name, customer address, account number, department number of sale, unit amount of sale, dollar amount of sale, previous balance, plus other pertinent information.

The basic subdivisions of a record (that is, those fields not further subdivided) are called **elementary items**. Thus, a record can be made up of a series of elementary items, or it can itself be an elementary item.

It might be necessary to refer to a set of elementary items; thus, elementary items can be combined into **group items**. Groups themselves can be combined into a more inclusive group that contains one or more subgroups. Thus, within one hierarchy of data items, an elementary item can belong to more than one group item.

A system of level-numbers specifies the organization of elementary and group items into records. Special level-numbers are also used; they identify data items used for special purposes.

## Data Relationships

### Levels of Data in a Record Description Entry

Each group and elementary item in a record requires a separate entry, and each must be assigned a level-number.

A level-number is a 1- or 2-digit integer between 01 and 49, or one of three special level-numbers: 66, 77, or 88. The following level-numbers are used to structure records:

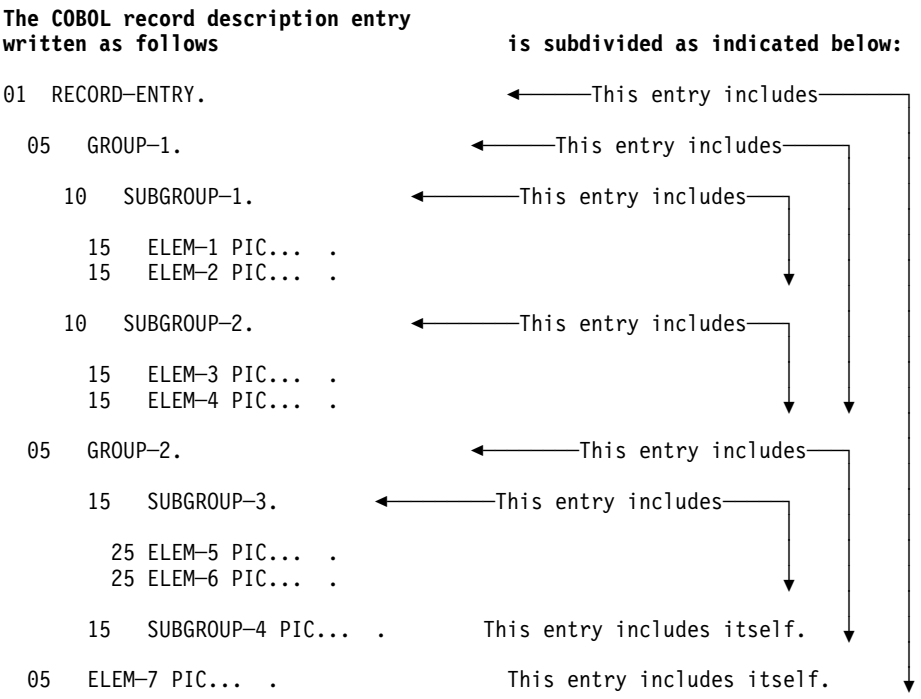
01 This level-number specifies the record itself, and is the most inclusive level-number possible. A level-01 entry can be either a group item or an elementary item. It must begin in Area A.

02-49
These level-numbers specify group and elementary items within a record. They can begin in Area A or Area B. Less inclusive data items are assigned higher (not necessarily consecutive) level-numbers in this series.

A group item includes all group and elementary items following it, until a level-number less than or equal to the level-number of this group is encountered.

All elementary or group items immediately subordinate to one group item must be assigned identical level-numbers higher than the level-number of this group item.

Figure 4 on page 135 illustrates the concept. Note that all groups immediately subordinate to the level-01 entry have the same level-number. Note also that elementary items from different subgroups do not necessarily have the same level numbers, and that elementary items can be specified at any level within the hierarchy.




The storage arrangement of the record description entry is illustrated below:

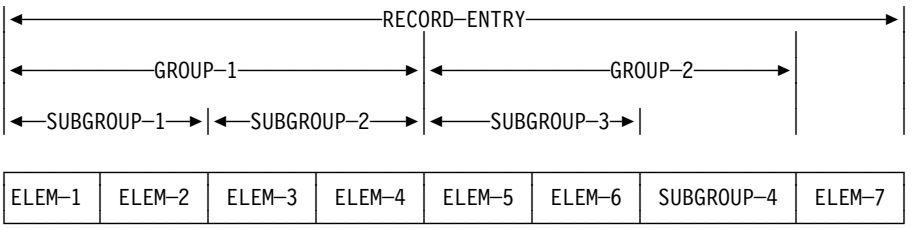



*Figure 4. Levels in a Record Description*

## Data Relationships

x IBM COBOL accepts nonstandard level-numbers that are not identical to others at the
x same level. For example, the following two record description entries are equivalent:

```
01   EMPLOYEE-RECORD.
     05  EMPLOYEE-NAME.
         10  FIRST-NAME PICTURE  X(10).
         10  LAST-NAME  PICTURE  X(10).
     05  EMPLOYEE-ADDRESS.
         10  STREET     PICTURE  X(10).
         10  CITY       PICTURE  X(10).
01   EMPLOYEE-RECORD.
     05  EMPLOYEE-NAME.
         10  FIRST-NAME PICTURE  X(10).
         10  LAST-NAME  PICTURE  X(10).
     04  EMPLOYEE-ADDRESS.
         08  STREET     PICTURE  X(10).
         08  CITY       PICTURE  X(10).
```

## Special Level-Numbers

Special level-numbers identify items that do not structure a record. The special level-numbers are:

66 Identifies items that must contain a RENAMES clause; such items regroup previously defined data items.

(For details, see "RENAMES Clause" on page 190.)

77 Identifies data item description entries — independent Working-Storage or Linkage Section items that are not subdivisions of other items, and are not subdivided themselves. Level-77 items must begin in Area A.

88 Identifies any condition-name entry that is associated with a particular value of a conditional variable. (For details, see "VALUE Clause" on page 207.)

**Note:** Level-77 and level-01 entries in the Working-Storage and Linkage Sections that are referenced in the program must be given unique data-names, because neither can be qualified. Subordinate data-names that are referenced in the program must be either uniquely defined, or made unique through qualification. Unreferenced data-names need not be uniquely defined.

## Indentation

Successive data description entries can begin in the same column as preceding entries, or can be indented. Indentation is useful for documentation, but does not affect the action of the compiler.

## Classes and Categories of Data

All data used in a COBOL program can be divided into classes and categories.

Every group item belongs to the alphanumeric class, even if the subordinate elementary items belong to another class.

## Data Relationships

Every elementary item in a program belongs to one of the classes as well as to one of the categories. Table 11 on page 137 shows the relationship among data classes and categories.

Every data item which is a function is an elementary item, and belongs to the category alphanumeric or numeric, and to the corresponding class; the category of each function is determined by the definition of the function.

*Table 11. Classes and Categories of Data*

| | Level of Item | Class | Category |
|---|---|---|---|
| | Elementary | Alphabetic | Alphabetic |
| | | Numeric | Numeric |
| x | | | Internal Floating-point |
| x | | | External Floating-point |
| | | Alphanumeric | Numeric-Edited |
| | | | Alphanumeric-Edited |
| | | | Alphanumeric |
| x | | | DBCS |
| | Group | Alphanumeric | Alphabetic |
| | | | Numeric |
| x | | | Internal Floating-point |
| x | | | External Floating-point |
| | | | Numeric-Edited |
| | | | Alphanumeric-Edited |
| | | | Alphanumeric |
| x | | | DBCS |

## Alignment Rules

The standard alignment rules for positioning data in an elementary item depend on the category of a receiving item (that is, an item into which the data is moved; see "Elementary Moves" on page 334).

**Numeric**

For such receiving items, the following rules apply:

1. The data is aligned on the assumed decimal point and, if necessary, truncated or padded with zeros. (An **assumed decimal point** is one that has logical meaning but that does not exist as an actual character in the data.)
2. If an assumed decimal point is not explicitly specified, the receiving item is treated as though an assumed decimal point is specified immediately to the right of the field. The data is then treated according to the preceding rule.

## Data Relationships

**Numeric-edited**
The data is aligned on the decimal point, and (if necessary) truncated or padded with zeros at either end, except when editing causes replacement of leading zeros.

x **Internal Floating-point**
x A decimal point is assumed immediately to the left of the field. The data is
x aligned then on the leftmost digit position following the decimal point, with
x the exponent adjusted accordingly.

x **External Floating-point**
x The data is aligned on the leftmost digit position; the exponent is adjusted
x accordingly.

**Alphanumeric, Alphanumeric-Edited, Alphabetic, DBCS**
For these receiving items, the following rules apply:

1. The data is aligned at the leftmost character position, and (if necessary) truncated or padded with spaces at the right.
2. If the JUSTIFIED clause is specified for this receiving item, the above rule is modified, as described in "JUSTIFIED Clause" on page 162.

WRKSTN Under AIX, OS/2, and Windows, using control characters X'00' through X'1F' within an alphanumeric literal can give unpredictable results, which are not diagnosed by the compiler. Use hex literals instead.

## Standard Data Format

COBOL makes data description as machine independent as possible. For this reason, the properties of the data are described in relation to a standard data format rather than a machine-oriented format.

The standard data format uses the decimal system to represent numbers, no matter what base is used by the system, and uses all the characters of the character set of the computer to represent nonnumeric data.

## Character-String and Item Size

In your program, the size of an elementary item is determined through the number of character positions specified in its PICTURE character-string. In storage, however, the size is determined by the actual number of bytes the item occupies, as determined by the combination of its PICTURE character-string and its USAGE clause.

x For internal floating-point items, the size of the item in storage is determined by its
x USAGE clause. USAGE COMPUTATIONAL-1 reserves 4 bytes of storage for the item;
x USAGE COMPUTATIONAL-2 reserves 8 bytes of storage.

Normally, when an arithmetic item is moved from a longer field into a shorter one, the compiler truncates the data to the number of characters represented in the shorter item's PICTURE character-string.

For example, if a sending field with PICTURE S99999, and containing the value +12345, is moved to a BINARY receiving field with PICTURE S99, the data is truncated to +45. For additional information see “USAGE Clause” on page 200.

x The TRUNC compiler option can affect the value of a binary numeric item. For infor-
x mation on TRUNC, see the *IBM COBOL Programming Guide* for your platform.

## Signed Data

There are two categories of algebraic signs used in IBM COBOL: operational signs and editing signs.

## Operational Signs

Operational signs are associated with signed numeric items, and indicate their algebraic properties. The internal representation of an algebraic sign depends on the item's USAGE clause, its SIGN clause (if present), and on the operating environment involved. (For further details about the internal representation see “USAGE Clause” on page 200.) Zero is considered a unique value, regardless of the operational sign. An unsigned field is always assumed to be either positive or zero.

## Editing Signs

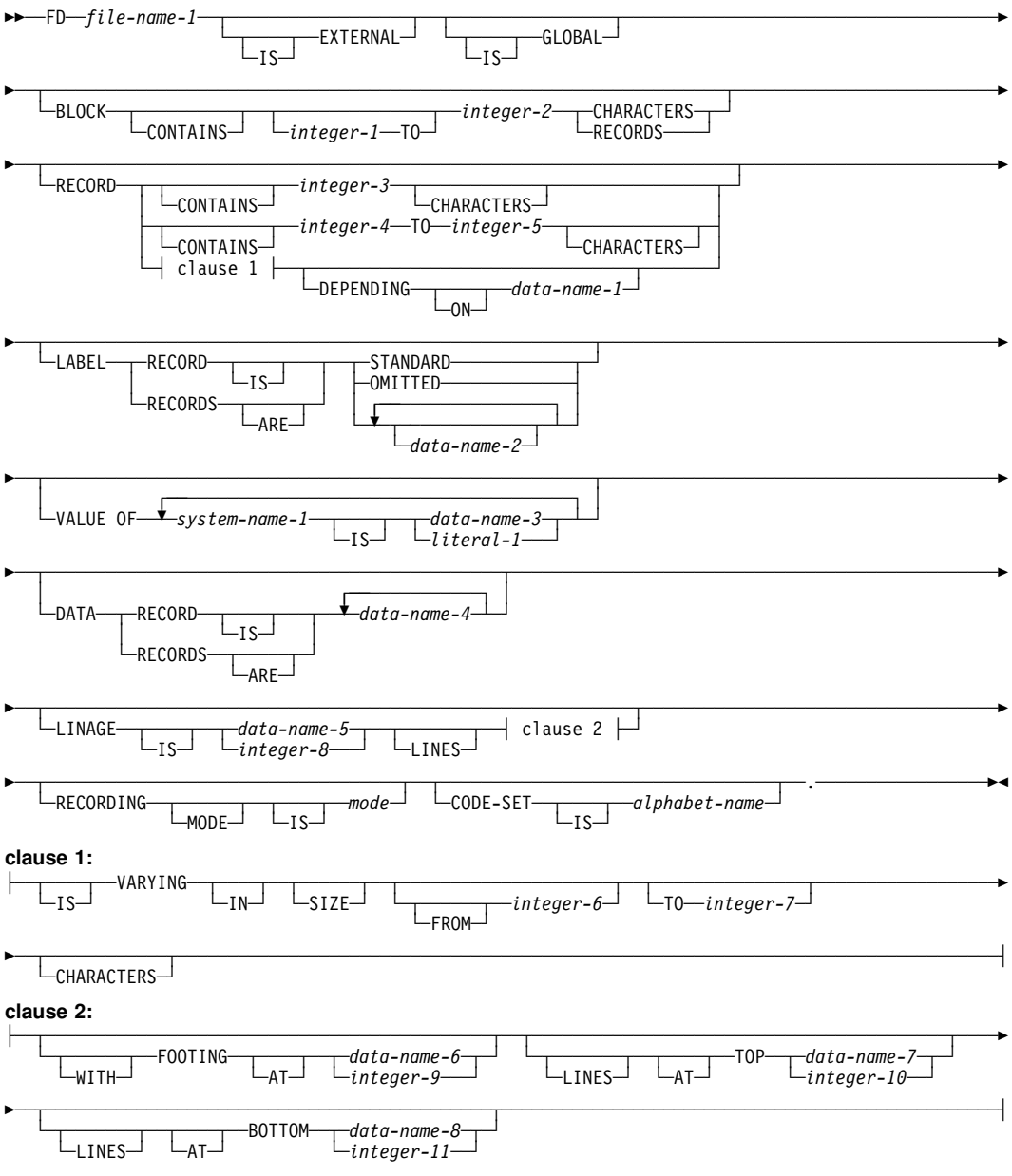
Editing signs are associated with numeric-edited items; editing signs are PICTURE symbols that identify the sign of the item in edited output.

# Data Division—File Description Entries

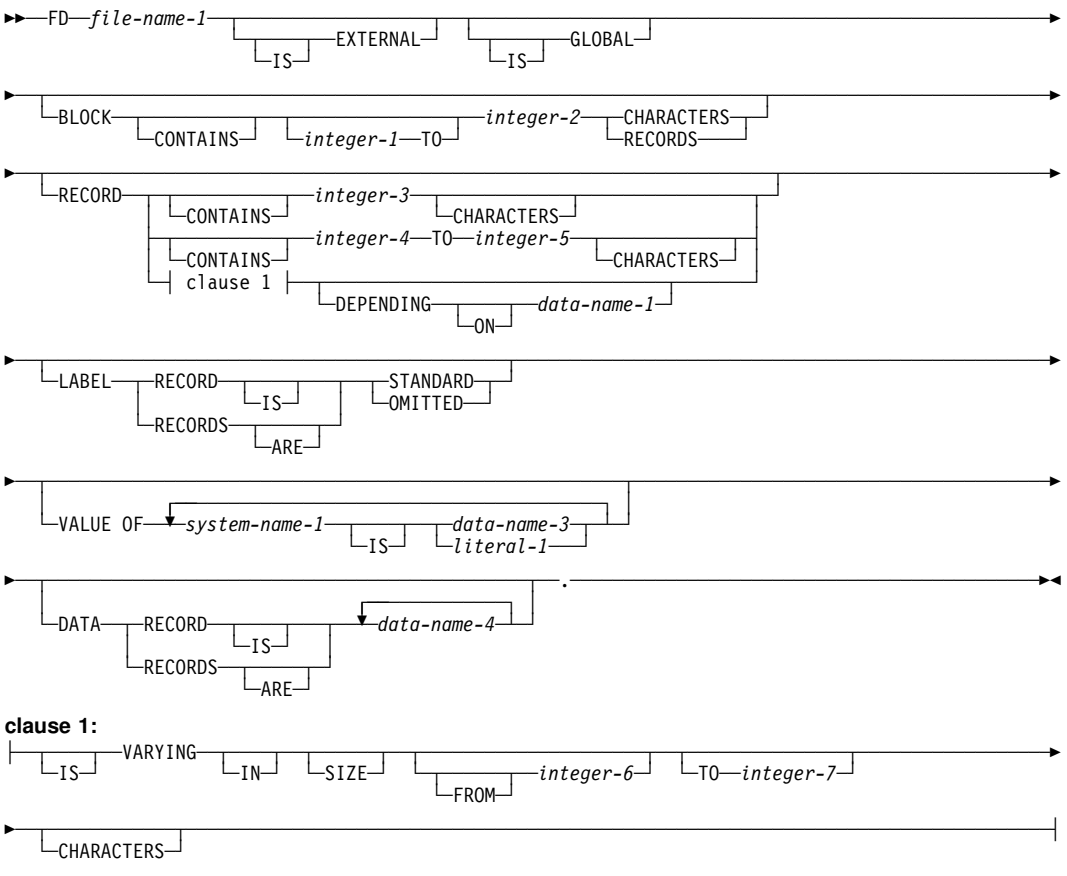
In a COBOL program, the **File Description (FD) Entry** (or **Sort File Description (SD) Entry** for sort/merge files) represents the highest level of organization in the File Section. The order in which the optional clauses follow the FD or SD entry is not important.


© Copyright IBM Corp. 1991, 1996

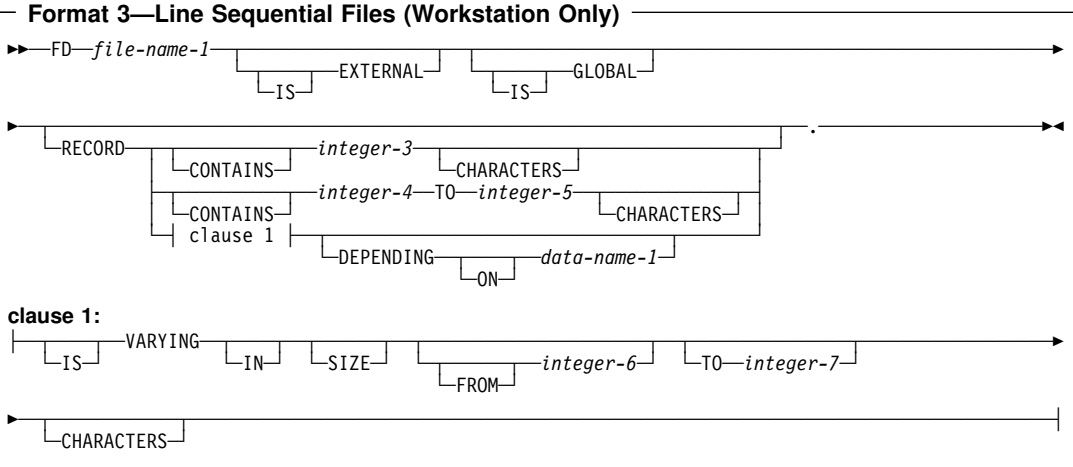
**Format 1—Sequential Files**



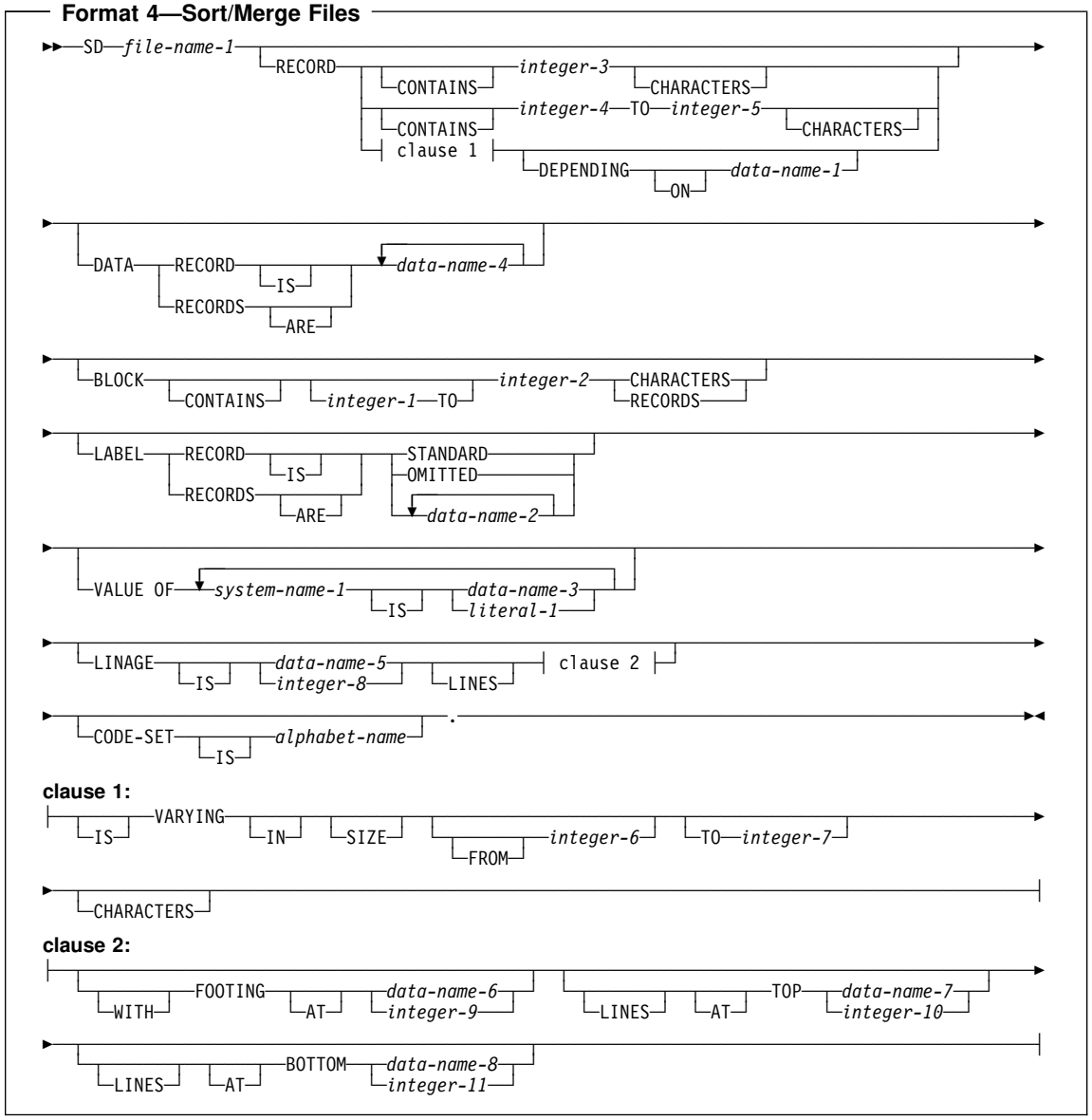
**Format 2—Relative/Indexed Files**




**Format 3—Line Sequential Files (Workstation Only)**



**Format 4—Sort/Merge Files**



## File Section

The File Section must contain a level indicator for each input and output file:

- For all files except sort/merge, the File Section must contain an FD entry.
- For each sort or merge file, the File Section must contain an SD entry.

**file-name**
Must follow the level indicator (FD or SD), and must be the same as that specified in the associated SELECT clause. The file-name must adhere to the rules of formation for a user-defined word; at least one character must be alphabetic. The file-name must be unique within this program.

One or more record description entries must follow the file-name. When more than one record description entry is specified, each entry implies a redefinition of the same storage area.

The clauses that follow file-name are optional; they can appear in any order.

**FD (Formats 1, 2, and 3)**
The last clause in the FD entry must be immediately followed by a separator period.

**SD (Format 4)**
An SD entry must be written for each sort or merge file in the program. The last clause in the SD entry must be immediately followed by a separator period.

The following example illustrates the File Section entries needed for a sort or merge file:

```
SD  SORT-FILE.

01  SORT-RECORD  PICTURE X(80).
```

## EXTERNAL Clause

The EXTERNAL clause specifies that a file connector is external, and permits communication between two programs by the sharing of files. A file connector is external if the storage associated with that file is associated with the run unit rather than with any particular program within the run unit. An external file can be referenced by any program in the run unit that describes the file. References to an external file from different programs using separate descriptions of the file are always to the same file. In a run unit, there is only one representative of an external file.

In the File Section, the EXTERNAL clause can only be specified in file description entries.

The records appearing in the file description entry need not have the same name in corresponding external file description entries. In addition, the number of such records need not be the same in corresponding file description entries.

Use of the EXTERNAL clause does not imply that the associated file-name is a global name. See the *IBM COBOL Programming Guide* for your platform for specific information on the use of the EXTERNAL clause.

## GLOBAL Clause

The GLOBAL clause specifies that the file connector named by a file-name is a global name. A global file-name is available to the program that declares it and to every program that is contained directly or indirectly in that program.

A file-name is global if the GLOBAL clause is specified in the file description entry for that file-name. A record-name is global if the GLOBAL clause is specified in the record description entry by which the record-name is declared or, in the case of record description entries in the File Section, if the GLOBAL clause is specified in the file description entry for the file-name associated with the record description entry. (See the *IBM COBOL Programming Guide* for your platform for specific information on the use of the GLOBAL clause.)

Two programs in a run unit can reference global file connectors in the following circumstances:

1. An external file connector can be referenced from any program that describes that file connector.
2. If a program is contained within another program, both programs can refer to a global 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.

## BLOCK CONTAINS Clause

x WRKSTN Under AIX, OS/2, and Windows, the BLOCK CONTAINS clause is not sup-
x ported for line sequential files. It is treated as a comment for sequential, relative and indexed files.

The BLOCK CONTAINS clause specifies the size of the physical records. The characters in the BLOCK CONTAINS clause reflect the number of bytes in the record. (For
x example, if you have a block with 10 DBCS characters, the BLOCK CONTAINS clause should say `BLOCK CONTAINS 20 CHARACTERS`.

If the records in the file are not blocked, the BLOCK CONTAINS clause can be omitted. When it is omitted, the compiler assumes that records are not blocked. Even if each physical record contains only one complete logical record, coding BLOCK CONTAINS 1 RECORD would result in fixed blocked records.

The BLOCK CONTAINS clause can be omitted when the associated File Control entry specifies a VSAM file; the concept of blocking has no meaning for VSAM files; the clause is syntax checked, but it has no effect on the execution of the program.

## BLOCK CONTAINS Clause

For EXTERNAL files, the value of all BLOCK CONTAINS clauses of corresponding EXTERNAL files must match within the run unit. This conformance is in terms of character positions and does not depend upon whether the value was specified as CHARACTERS or as RECORDS.

**integer-1, integer-2**
Must be nonzero unsigned integers. They specify the number of:

**CHARACTERS**
Specifies the number of character positions required to store the physical record, no matter what USAGE the characters have within the data record.

If only integer-2 is specified, it specifies the exact character size of the physical record. When integer-1 and integer-2 are both specified, they represent, respectively, the minimum and maximum character sizes of the physical record.

Integer-1 and integer-2 must include any control bytes and padding contained in the physical record. (Logical records do not include padding.)

The CHARACTERS phrase is the default. CHARACTERS must be specified when:

- The physical record contains padding.
- Logical records are grouped so that an inaccurate physical record size could be implied. For example, suppose you describe a variable-length record of 100 characters, yet each time you write a block of 4, one 50-character record is written followed by three 100-character records. If the RECORDS phrase were specified, the compiler would calculate the block size as 420 characters instead of the actual size, 370 characters. (This calculation includes block and record descriptors.)

**RECORDS**
Specifies the number of logical records contained in each physical record.

The compiler assumes that the block size must provide for integer-2 records of maximum size, and provides any additional space needed for control bytes.

x When running under MVS, BLOCK CONTAINS 0 can be specified for QSAM files; the
x block size is determined at object time from the DD parameters or the data set label.

x If the RECORD CONTAINS 0 CHARACTERS clause is specified, and the BLOCK
x CONTAINS 0 CHARACTERS clause is specified (or omitted), the block size is deter-
x mined at object time from the DD parameters or the data set label of the file. For
x output data sets, with either of the above conditions, the DCB used by Language Envi-
x ronment will have a zero block size value. If you do not specify a block size value, the
x operating system might select a System Determined Block Size (SDB). See the oper-
x ating system specifications for further information on SDB.

x BLOCK CONTAINS can be omitted for SYSIN/SYSOUT files under MVS. The blocking
x is determined by the operating system.

When running under CMS, BLOCK CONTAINS 0 can be specified for QSAM files; the block size is determined at object time from the FILEDEF parameters or the data set label. If the RECORD CONTAINS 0 CHARACTERS clause is specified, and the BLOCK CONTAINS clause is omitted (or if the BLOCK CONTAINS 0 CHARACTERS clause is specified), the block size is determined at object time from the FILEDEF parameters or the data set label of the file.

Under VM, the BLOCK CONTAINS 0 clause might cause blocked or unblocked records to be used for an output file, depending on the FILEDEF options specified. The DCB used by Language Environment will have a zero block size, so the FILEDEF uses the CMS defaults. The defaults are documented in the *CMS Command Reference*, under the FILEDEF command.

The BLOCK CONTAINS clause is treated as a comment under an SD.

The BLOCK CONTAINS clause cannot be used with the RECORDING MODE U clause.

## RECORD Clause

When the RECORD clause is used, the record size must be specified as the number of character positions needed to store the record internally. That is, it must specify the number of bytes occupied internally by the characters of the record (not the number of characters used to represent the item within the record). (For example, if you have a record with 10 DBCS characters, the RECORD clause should say `RECORD CONTAINS 20 CHARACTERS`.)

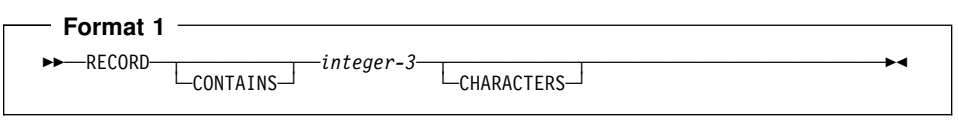
The size of a record is determined according to the rules for obtaining the size of a group item. (See “USAGE Clause” on page 200 and “SYNCHRONIZED Clause” on page 193.)

When the RECORD clause is omitted, the compiler determines the record lengths from the record descriptions. When one of the entries within a record description contains an OCCURS DEPENDING ON clause, the compiler uses the maximum value of the variable-length item to calculate the number of character positions needed to store the record internally.

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 must specify the same maximum number of character positions.

### Format 1

Format 1 specifies the number of character positions for fixed-length records.



## RECORD Clause

**integer-3**
Must be an unsigned integer that specifies the number of character positions contained in each record in the file.

WRKSTN Under AIX, OS/2, and Windows the RECORD CONTAINS clause is valid. However, the RECORD CONTAINS 0 CHARACTERS clause is not supported for line sequential files. It is treated as a comment for sequential, relative and indexed files.

MVS/VM Under MVS, the RECORD CONTAINS 0 CHARACTERS clause can be specified for input QSAM files containing fixed-length records; the record size is determined at object time from the DD statement parameters or the data set label. If, at object time, the actual record is larger than the 01 record description, only the 01 record length is available. If the actual record is shorter, only the actual record length can be referred to. Otherwise, uninitialized data or an addressing exception can be produced.

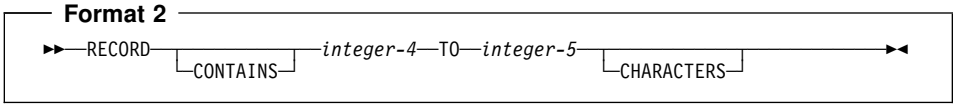
**Note:** If the RECORD CONTAINS 0 clause is specified, then the SAME AREA, SAME RECORD AREA, or APPLY WRITE-ONLY clauses cannot be specified.

Do not specify the RECORD CONTAINS 0 clause for an SD entry.

## Format 2

Format 2 specifies the number of character positions for either fixed-length or variable-length records. Fixed-length records are obtained when all 01 record description entry lengths are the same. The Format 2 RECORD CONTAINS clause is never required, because the minimum and maximum record lengths are determined from the record description entries.

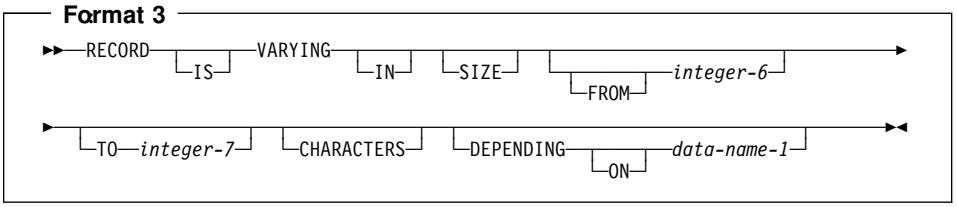
WRKSTN Under AIX, OS/2, and Windows, the Format 2 RECORD clause is not supported for line sequential files.




**integer-4**
**integer-5**
Must be unsigned integers. Integer-4 specifies the size of the smallest data record, and integer-5 specifies the size of the largest data record.

## Format 3

Format 3 is used to specify variable-length records.




**integer-6**
Specifies the minimum number of character positions to be contained in any record of the file. If integer-6 is not specified, the minimum number of character positions to be contained in any record of the file is equal to the least number of character positions described for a record in that file.

**integer-7**
Specifies the maximum number of character positions in any record of the file. If integer-7 is not specified, the maximum number of character positions to be contained in any record of the file is equal to the greatest number of character positions described for a record in that file.

The number of character positions associated with a record description is determined by the sum of the number of character positions in all elementary data items (excluding redefinitions and renamings), plus any implicit FILLER due to synchronization. If a table is specified:

- The minimum number of table elements described in the record is used in the summation above to determine the minimum number of character positions associated with the record description.
- The maximum number of table elements described in the record is used in the summation above to determine the maximum number of character positions associated with the record description.

If data-name-1 is specified:

- Data-name-1 must be an elementary unsigned integer.
- The number of character positions in the record must be placed into the data item referenced by data-name-1 before any RELEASE, REWRITE, or WRITE statement is executed for the file.
- 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.
- After the successful execution of a READ or RETURN statement for the file, the contents of the data item referenced by data-name-1 indicate the number of character positions in the record just read.

## LABEL RECORDS Clause

During the execution of a RELEASE, REWRITE, or WRITE statement, the number of character positions in the record is determined by the following conditions:

- If data-name-1 is specified, by the content of the data item referenced by data-name-1.
- If data-name-1 is not specified and the record does not contain a variable occurrence data item, by the number of character positions in the record.
- If data-name-1 is not specified and the record contains a variable occurrence data item, by the sum of the fixed position and that portion of the table described by the number of occurrences at the time of execution of the output statement.

During the execution of a READ ... INTO or RETURN ... INTO statement, the number of character positions in the current record that participate as the sending data items in the implicit MOVE statement is determined by the following conditions:

- If data-name-1 is specified, by the content of the data item referenced by data-name-1.
- 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.

---

## LABEL RECORDS Clause

x WRKSTN Under AIX, OS/2, and Windows, the LABEL RECORDS clause is not sup-
x ported for line sequential files.

It is treated as a comment for sequential, relative and indexed files. A warning message is issued if you use any of the following language elements:

- LABEL RECORD IS data-name
- USE...AFTER...LABEL PROCEDURE
- GO TO MORE-LABELS

The LABEL RECORDS clause indicates the presence or absence of labels. If it is not specified for a file, label records for that file must conform to the system label specifications.

For VSAM files, the LABEL RECORDS clause is syntax checked, but it has no effect on the execution of the program. COBOL label processing, therefore, is not performed.

**STANDARD**
Labels conforming to system specifications exist for this file.

STANDARD is permitted for mass storage devices and tape devices.

**OMITTED**
No labels exist for this file.

OMITTED is permitted for tape devices.

x **data-name-2**
x User labels are present in addition to standard labels. Data-name-2 specifies the
x name of a user label record. Data-name-2 must appear as the subject of a record
x description entry associated with the file.

x The LABEL RECORDS clause is treated as a comment under an SD.

## VALUE OF Clause

x WRKSTN Under AIX, OS/2, and Windows, the VALUE OF clause is not supported for
x line sequential files. It is treated as a comment for sequential, relative and indexed files.

The VALUE OF clause describes an item in the label records associated with this file. The clause is syntax checked, but has no effect on the execution of the program.

**data-name-3**
Should be qualified when necessary, but cannot be subscripted. It must be described in the Working-Storage Section. It cannot be described with the USAGE IS INDEX clause.

**literal-1**
Can be numeric or nonnumeric, or a figurative constant of category numeric or nonnumeric.

Cannot be a floating-point literal.

x The VALUE OF clause is treated as a comment under an SD.

## DATA RECORDS Clause

x WRKSTN Under AIX, OS/2, and Windows, the DATA RECORDS clause is not sup-
x ported for line sequential files. It is treated as a comment for sequential, relative and indexed files.

The DATA RECORDS clause is syntax checked, but it serves only as documentation for the names of data records associated with this file.

**data-name-4**
The names of record description entries associated with this file.

x As an IBM extension, the data-name need not have an 01 level number record
x description with the same name associated with it.

## LINAGE Clause

The LINAGE clause specifies the depth of a logical page in terms of number of lines. Optionally, it also specifies the line number at which the footing area begins, as well as the top and bottom margins of the logical page. (The logical page and the physical page cannot be the same size.)

## LINAGE Clause

The LINAGE clause is effective for sequential and line sequential files opened OUTPUT
x and, as an IBM extension, EXTEND.

All integers must be unsigned. All data-names must be described as unsigned integer data items.

**data-name-5**
**integer-8**
The number of lines that can be written and/or spaced on this logical page. The area of the page that these lines represent is called the **page body**. The value must be greater than zero.

**WITH FOOTING AT**
Integer-9 or the value of the data item in data-name-6 specifies the first line number of the footing area within the page body. The footing line number must be greater than zero, and not greater than the last line of the page body. The footing area extends between those two lines.

**LINES AT TOP**
Integer-10 or the value of the data item in data-name-7 specifies the number of lines in the top margin of the logical page. The value can be zero.

**LINES AT BOTTOM**
Integer-11 or the value of the data item in data-name-8 specifies the number of lines in the bottom margin of the logical page. The value can be zero.

Figure 5 illustrates the use of each phrase of the LINAGE clause.

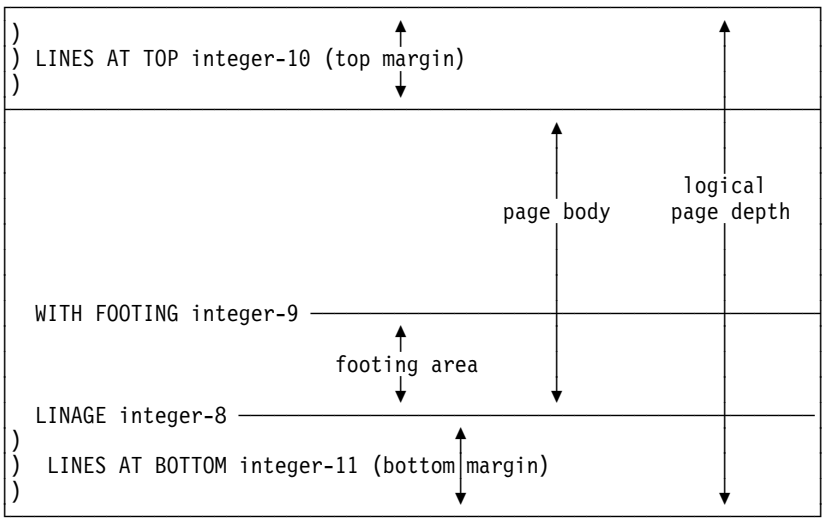



*Figure 5. LINAGE Clause Phrases*

The logical page size specified in the LINAGE clause is the sum of all values specified in each phrase except the FOOTING phrase. If the LINES AT TOP and/or the LINES

AT BOTTOM phrase is omitted, the assumed value for top and bottom margins is zero. Each logical page immediately follows the preceding logical page, with no additional spacing provided.

If the FOOTING phrase is omitted, its assumed value is equal to that of the page body (integer-8 or data-name-5).

At the time an OPEN OUTPUT statement is executed, the values of integer-8, integer-9, integer-10, and integer-11, if specified, are used to determine the page body, first footing line, top margin, and bottom margin of the logical page for this file. See Figure 5 on page 152 above. These values are then used for all logical pages printed for this file during a given execution of the program.

At the time an OPEN statement with the OUTPUT phrase is executed for the file, data-name-5, data-name-6, data-name-7, and data-name-8 determine the page body, first footing line, top margin, and bottom margin for the first logical page only.

At the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, the values of data-name-5, data-name-6, data-name-7, and data-name-8 if specified, are used to determine the page body, first footing line, top margin, and bottom margin for the next logical page.

If an external file connector is associated with this file description entry, all file description entries in the run unit that are associated with this file connector must have:

- A LINAGE clause, if any file description entry has a LINAGE clause.
- The same corresponding values for integer-8, integer-9, integer-10, and integer-11, if specified.
- The same corresponding external data items referenced by data-name-5, data-name-6, data-name-7, and data-name-8.

See “ADVANCING Phrase” on page 417 for the behavior of carriage control characters in EXTERNAL files.

x The LINAGE clause is treated as a comment under an SD.

## LINAGE-COUNTER Special Register

For information about the LINAGE-COUNTER Special Register, see “LINAGE-COUNTER” on page 16.

## RECORDING MODE Clause

### Under MVS and VM

The RECORDING MODE clause specifies the format of the physical records in a QSAM file. The clause is ignored for a VSAM file.

Permitted values for RECORDING MODE are:

**Recording Mode F (Fixed)**
All the records in a file are the same length and each is wholly contained within one block. Blocks can contain more than one record, and there is usually a fixed number of records for each block. In this mode, there are no record-length or block-descriptor fields.

**Recording Mode V (Variable)**
The records can be either fixed-length or variable-length, and each must be wholly contained within one block. Blocks can contain more than one record. Each data record includes a record-length field and each block includes a block-descriptor field. These fields are not described in the Data Division. They are each 4 bytes long and provision is automatically made for them. These fields are not available to you.

**Recording Mode U (Fixed or Variable)**
The records can be either fixed-length or variable-length. However, there is only one record for each block. There are no record-length or block-descriptor fields.

**Note:** You cannot use RECORDING MODE U if you are using the BLOCK CONTAINS clause.

**Recording Mode S (Spanned)**
The records can be either fixed-length or variable-length, and can be larger than a block. If a record is larger than the remaining space in a block, a segment of the record is written to fill the block. The remainder of the record is stored in the next block (or blocks, if required). Only complete records are made available to you. Each segment of a record in a block, even if it is the entire record, includes a segment-descriptor field, and each block includes a block-descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to you.

**Note:** When recording mode S is used, the BLOCK CONTAINS CHARACTERS clause must be used.

Recording mode S is not allowed for ASCII files.

If the RECORDING MODE clause is not specified for a QSAM file, the IBM COBOL for MVS & VM compiler determines the recording mode as follows:

**F** The compiler determines the recording mode to be F if the largest level-01 record associated with the file is not greater than the block size specified in the BLOCK CONTAINS clause, and you do one of the following:

- Use the RECORD CONTAINS *integer* clause (for more information, see *IBM COBOL for MVS & VM Compiler and Run-Time Migration Guide*.)
- Omit the RECORD clause and make sure all level-01 records associated with the file are the same size and none contain an OCCURS DEPENDING ON clause.

**V** The compiler determines the recording mode to be V if the largest level-01 record associated with the file is not greater than the block size specified in the BLOCK CONTAINS clause, and you do one of the following:

- Use the RECORD IS VARYING clause
- Omit the RECORD clause and make sure all level-01 records associated with the file are not the same size or some contain an OCCURS DEPENDING ON clause
- Use the RECORD CONTAINS *integer-1* TO *integer-2* clause with integer-1 the minimum length and integer-2 the maximum length of the level-01 records associated with the file. The two integers must be different, with values matching minimum and maximum length of either different length records or record(s) with an OCCURS DEPENDING ON clause.

**S** The compiler determines the recording mode to be S if the maximum block size is smaller than the largest record size.

**U** Recording mode U is never obtained by default. The RECORDING MODE U clause must be explicitly used.

## Under AIX, OS/2, and Windows

Under AIX, OS/2, and Windows, the RECORDING MODE clause is not supported for line sequential files. It is treated as a comment for a relative or indexed file. For record sequential files, the RECORDING MODE clause is treated as follows:

**F** Record descriptions are validated as fixed. Do not specify RECORDING MODE F if the record descriptions are variable.

**V** Variable length record format is assumed (even if the record descriptions are fixed).

**U** Treated as a comment.

**S** Treated the same as V.

## CODE-SET Clause

x WRKSTN Under AIX, OS/2, and Windows the CODE-SET clause is not supported for
x line sequential files. It is treated as a comment for sequential, relative and indexed files.

The CODE-SET clause specifies the character code used to represent data on a magnetic tape file. When the CODE-SET clause is specified, an alphabet-name identifies the character code convention used to represent data on the input-output device.

Alphabet-name must be defined in the SPECIAL-NAMES paragraph as STANDARD-1 (for ASCII-encoded files), as STANDARD-2 (for ISO 7-bit encoded files), as EBCDIC (for EBCDIC-encoded files), or as NATIVE. When NATIVE is specified, the CODE-SET clause is syntax checked, but it has no effect on the execution of the program.

The CODE-SET clause also specifies the algorithm for converting the character codes on the input-output medium from/to the internal EBCDIC character set.

When the CODE-SET clause is specified for a file, all data in this file must have USAGE DISPLAY, and, if signed numeric data is present, it must be described with the SIGN IS SEPARATE clause.

When the CODE-SET clause is omitted, the EBCDIC character set is assumed for this file.

If the associated file connector is an external file connector, all CODE-SET clauses in the run unit that are associated with that file connector must have the same character set.

The CODE-SET clause is valid only for magnetic tape files.

x The CODE-SET clause is treated as a comment under an SD.

# Data Division—Data Description Entry

A data description entry specifies the characteristics of a data item.

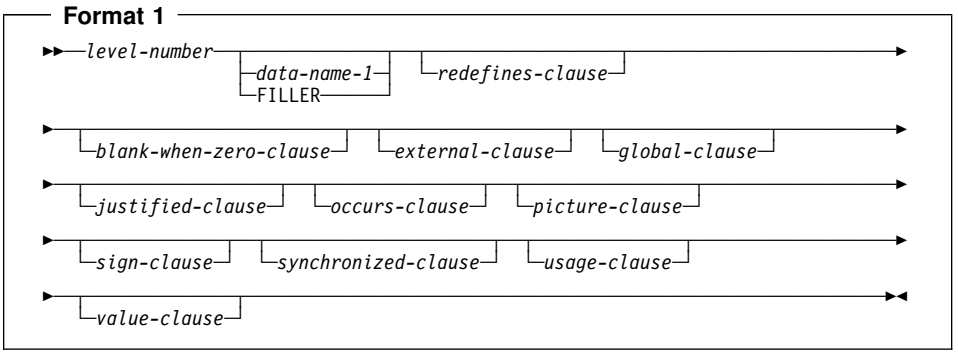
This chapter describes the coding of data description entries and record description entries (which are sets of data description entries). The single term **data description entry** is used in this chapter to refer to data and record description entries.

Data description entries that define **independent** data items do not make up a record. These are known as **data item description entries**.

The data description entry has three general formats. All data description entries must end with a separator period.

## Format 1

Format 1 is used for data description entries in all Data Division sections.




**Note:** The clauses can be written in any order with two exceptions:

If data-name or FILLER is specified, it must immediately follow the level-number.

When the REDEFINES clause is specified, it must immediately follow data-name or FILLER, if either is specified. If data-name or FILLER is not specified, the REDEFINES clause must immediately follow the level-number.

Level-number in Format 1 can be any number from 01-49 or 77.

A space, a separator comma, or a separator semicolon must separate clauses.

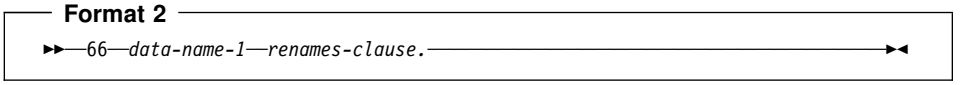
© Copyright IBM Corp. 1991, 1996

## Level-Numbers

### Format 2

Format 2 regroups previously defined items.

**Format 2**




A level-66 entry cannot rename another level-66 entry, nor can it rename a level-01, level-77, or level-88 entry.

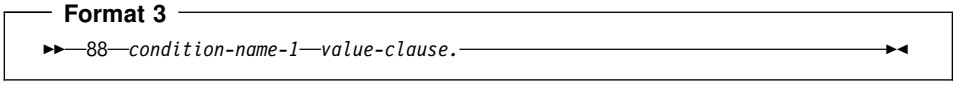
All level-66 entries associated with one record must immediately follow the last data description entry in that record.

Details are contained in “RENAMES Clause” on page 190.

### Format 3

Format 3 describes condition-names.

**Format 3**




**condition-name**
A user-specified name that associates a value, a set of values, or a range of values with a conditional variable.

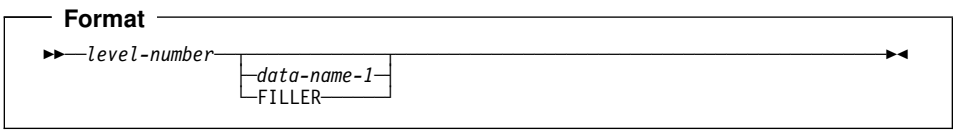
A **conditional variable** is a data item that can assume one or more values, that can, in turn, be associated with a condition-name.

Format 3 can be used to describe both elementary and group items. Further information on condition-name entries can be found under “VALUE Clause” on page 207.

## Level-Numbers

The level-number specifies the hierarchy of data within a record, and identifies special-purpose data entries. A level-number begins a data description entry, a renamed or redefined item, or a condition-name entry. A level-number has a value taken from the set of integers between 1 and 49, or from one of the special level-numbers, 66, 77, or 88.

**Format**



## Level-Numbers

**level-number**

01 and 77 must begin in Area A and must be followed either by a separator period; or by a space, followed by its associated data-name, FILLER, or appropriate data description clause.

Level numbers 02 through 49 can begin in Areas A or B and must be followed by a space or a separator period.

Level numbers 66 and 88 can begin in Areas A or B and must be followed by a space.

Single-digit level-numbers 1 through 9 can be substituted for level-numbers 01 through 09.

Successive data description entries can start in the same column as the first or they can be indented according to the level-number. Indentation does not affect the magnitude of a level-number.

When level-numbers are indented, each new level-number can begin any number of spaces to the right of Area A. The extent of indentation to the right is limited only by the width of Area B.

For more information, see “Levels of Data” on page 133

**data-name**

Explicitly identifies the data being described.

If specified, a data-name identifies a data item used in the program. The data-name must be the first word following the level-number.

The data item can be changed during program execution.

Data-name must be specified for level-66 and level-88 items. It must also be specified for any entry containing the GLOBAL or EXTERNAL clause, and for record description entries associated with file description entries having the GLOBAL or EXTERNAL clauses.

**FILLER**

Is a data item that is not explicitly referred to in a program. The key word FILLER is optional. If specified, FILLER must be the first word following the level-number.

The key word FILLER can be used with a conditional variable, if explicit reference is never made to the conditional variable but only to values it can assume. FILLER cannot be used with a condition-name.

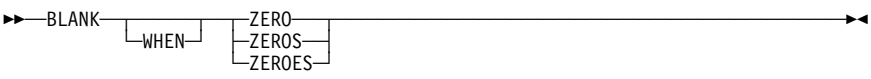
In a MOVE CORRESPONDING statement, or in an ADD CORRESPONDING or SUBTRACT CORRESPONDING statement, FILLER items are ignored. In an INITIALIZE statement, elementary FILLER items are ignored.

If the data-name or FILLER clause is omitted, the data item being described is treated as though FILLER had been specified.

## BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause specifies that an item contains nothing but spaces when its value is zero.

**Format**




The BLANK WHEN ZERO clause can be specified only for elementary numeric or numeric-edited items. These items must be described, either implicitly or explicitly, as USAGE IS DISPLAY. When the BLANK WHEN ZERO clause is specified for a numeric item, the item is considered a numeric-edited item.

The BLANK WHEN ZERO clause must not be specified for level-66 or level-88 items.

The BLANK WHEN ZERO clause must not be specified for the same entry as the PICTURE symbols S or *.

The BLANK WHEN ZERO clause is not allowed for:

- Items described with the USAGE IS INDEX clause
- DBCS items
- External or internal floating-point items
- Items described with USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE

## EXTERNAL Clause

The EXTERNAL clause specifies that the storage associated with a data item is associated with the run unit rather than with any particular program or method within the run unit. An external data item can be referenced by any program or method in the run unit that describes the data item. References to an external data item from different programs or methods using separate descriptions of the data item are always to the same data item. In a run unit, there is only one representative of an external data item.

The EXTERNAL clause can be specified only in data description entries whose level-number is 01. It can only be specified on data description entries that are in the Working-Storage Section of a program or method. It cannot be specified in Linkage Section or File Section data description entries. Any data item described by a data description entry subordinate to an entry describing an external record also attains the EXTERNAL attribute. Indexes in an external data record do not possess the external attribute.

The data contained in the record named by the data-name clause is external and can be accessed and processed by any program or method in the run unit that describes and, optionally, redefines it. This data is subject to the following rules:

- If two or more programs or methods within a run unit describe the same external data record, each record-name of the associated record description entries must be the same and the records must define the same number of standard data format characters. However, a program or method that describes an external record can 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 programs or methods in the run unit.
- Use of the EXTERNAL clause does not imply that the associated data-name is a global name.

## GLOBAL Clause

The GLOBAL clause specifies that a data-name is available to every program contained within the program that declares it, as long as the contained program does not itself have a declaration for that name. All data-names subordinate to or condition-names or indexes associated with a global name are global names.

A data-name is global if the GLOBAL clause is specified either in the data description entry by which the data-name is declared or in another entry to which that data description entry is subordinate. The GLOBAL clause can be specified in the Working-
x Storage Section, the File Section, the Linkage Section, and the Local-Storage Section, but only in data description entries whose level-number is 01.

In the same Data Division, the data description entries for any two data items for which the same data-name is specified must not include the GLOBAL clause.

A statement in a program contained directly or indirectly within a program which describes a global name can reference that name without describing it again.

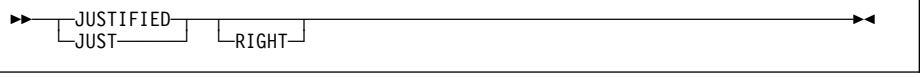
Two programs in a run unit can reference common data in the following circumstances:

1. The data content of an external data record can be referenced from any program provided that program has described that data record.
2. If a program is contained within another program, both programs can refer to data possessing the global attribute either in the containing program or in any program that directly or indirectly contains the containing program.

## JUSTIFIED Clause

The JUSTIFIED clause overrides standard positioning rules for a receiving item of the alphabetic or alphanumeric categories.

**Format**




You can only specify the JUSTIFIED clause at the elementary level. JUST is an abbreviation for JUSTIFIED, and has the same meaning.

You cannot specify the JUSTIFIED clause:

- For numeric, numeric-edited, or alphanumeric-edited items
- In descriptions of items described with the USAGE IS INDEX clause
- For items described as USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE
- For external or internal floating-point items
- For an edited DBCS item
- With level-66 (RENAMES) and level-88 (condition-name) entries

When the JUSTIFIED clause is specified for a receiving item, the data is aligned at the rightmost character position in the receiving item. Also:

- If the sending item is larger than the receiving item, the leftmost characters are truncated.
- If the sending item is smaller than the receiving item, the unused character positions at the left are filled with spaces.

The JUSTIFIED clause can be specified for a DBCS item (except edited DBCS items). When JUSTIFIED is specified for a receiving item, the data is aligned on the rightmost character position. If the sending item is larger than the receiving item, extra characters are truncated on the left. If the sending item is smaller than the receiving item, any unused positions on the left are filled with DBCS blanks.

If you omit the JUSTIFIED clause, the rules for standard alignment are followed (see “Alignment Rules” on page 137).

The JUSTIFIED clause does not affect initial settings, as determined by the VALUE clause.

## OCCURS Clause

The Data Division clauses used for table handling are the OCCURS clause and USAGE IS INDEX clause. For the USAGE IS INDEX description, see “USAGE Clause” on page 200.

The OCCURS clause specifies tables whose elements can be referred to by indexing or subscripting. It also eliminates the need for separate entries for repeated data items.

x The OCCURS clause can be specified for a DBCS item.

x An item described as USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or
x USAGE IS OBJECT REFERENCE can contain an OCCURS clause, or be subordinate
x to an item declared with an OCCURS clause.

x The OCCURS clause can be specified for external or internal floating-point items.

Formats for the OCCURS clause are:

- “Fixed-Length Tables” on page 164
- “Variable-Length Tables” on page 166

The **subject** of an OCCURS clause is the data-name of the data item containing the OCCURS clause. Except for the OCCURS clause itself, data description clauses used with the subject apply to each occurrence of the item described.

Whenever the subject of an OCCURS clause or any data-item subordinate to it is referenced, it must be subscripted or indexed with the following exceptions:

- When the subject of the OCCURS clause is used as the subject of a SEARCH statement.
- When the subject or subordinate data item is the object of the ASCENDING/DESCENDING KEY clause.
- When the subordinate data item is the object of the REDEFINES clause.

When subscripted or indexed, the subject refers to one occurrence within the table.

When not subscripted or indexed, the subject represents the entire table.

The OCCURS clause cannot be specified in a data description entry that:

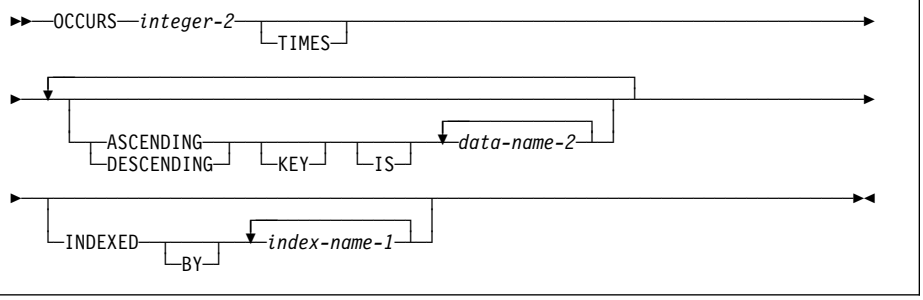
- Has a level number of 01, 66, 77, or 88.
- Describes a redefined data item. (However, a redefined item can be subordinate to an item containing an OCCURS clause.) See “REDEFINES Clause” on page 186.

## OCCURS Clause

### Fixed-Length Tables

Fixed-length tables are specified using the OCCURS clause. Because seven subscripts or indexes are allowed, six nested levels and one outermost level of the Format 1 OCCURS clause are allowed. The Format 1 OCCURS clause can be specified as subordinate to the OCCURS DEPENDING ON clause. In this way, a table of up to seven dimensions can be specified.

**Format 1—Fixed-Length Tables**




**integer-2**
The exact number of occurrences. Integer-2 must be greater than zero.

### ASCENDING/DESCENDING KEY Phrase

Data is arranged in ascending or descending order (depending on the key word specified) according to the values contained in data-name-2. The data-names are listed in their descending order of significance.

The order is determined by the rules for comparison of operands (see "Relation Condition" on page 229). The ASCENDING and DESCENDING KEY data items are used in OCCURS clauses and the SEARCH ALL statement for a binary search of the table element.

**data-name-2**
Must be the name of the subject entry, or the name of an entry subordinate to the subject entry. It can be qualified.

If data-name-2 names the subject entry, that entire entry becomes the ASCENDING/DESCENDING KEY, and is the only key that can be specified for this table element.

If data-name-2 does not name the subject entry, then data-name-2:

- Must be subordinate to the subject of the table entry itself
- Must **not** be subordinate to, or follow, any other entry that contains an OCCURS clause
- Must not contain an OCCURS clause.

Data-name-2 must not have subordinate items that contain OCCURS DEPENDING ON clauses.

When the ASCENDING/DESCENDING KEY phrase is specified, the following rules apply:

- Keys must be listed in decreasing order of significance.
- The total number of keys for a given table element must not exceed 12.
- You must arrange the data in the table in ASCENDING or DESCENDING sequence according to the collating sequence in use.
- A key can have DISPLAY, BINARY, PACKED-DECIMAL, or COMPUTATIONAL usage.
- The sum of the lengths of all the keys associated with one table element must not exceed 256.
- MVS/VM Under MVS and VM, a key can have COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, or COMPUTATIONAL-4 usage.
- WRKSTN Under AIX, OS/2, and Windows, a key can have COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, COMPUTATIONAL-4, or COMPUTATIONAL-5 usage.
- The ASCENDING/DESCENDING KEY phrase (for a SEARCH ALL statement only) can be specified in the OCCURS clause for a DBCS item.
- If a key is specified without qualifiers and it is not a unique name, the key will be implicitly qualified with the subject of the OCCURS clause and all qualifiers of the OCCURS clause subject.

The following example illustrates the specification of ASCENDING KEY data item:

```
WORKING-STORAGE SECTION.
01  TABLE-RECORD.
  05  EMPLOYEE-TABLE OCCURS 100 TIMES
      ASCENDING KEY IS WAGE-RATE EMPLOYEE-NO
      INDEXED BY A,  B.
    10  EMPLOYEE-NAME                        PIC X(20).
    10  EMPLOYEE-NO                          PIC 9(6).
    10  WAGE-RATE                            PIC 9999V99.
    10  WEEK-RECORD OCCURS 52 TIMES
        ASCENDING KEY IS WEEK-NO INDEXED BY C.
      15  WEEK-NO                            PIC 99.
      15  AUTHORIZED-ABSENCES                PIC  9.
      15  UNAUTHORIZED-ABSENCES              PIC  9.
      15  LATE-ARRIVALS                      PIC  9.
```

The keys for EMPLOYEE-TABLE are subordinate to that entry, while the key for WEEK-RECORD is subordinate to that subordinate entry.

In the preceding example, records in EMPLOYEE-TABLE must be arranged in ascending order of WAGE-RATE, and in ascending order of EMPLOYEE-NO within WAGE-RATE. Records in WEEK-RECORD must be arranged in ascending order of WEEK-NO. If they are not, results of any SEARCH ALL statement will be unpredictable.

## OCCURS Clause

## INDEXED BY Phrase

The INDEXED BY phrase specifies the indexes that can be used with a table. The INDEXED BY phrase is required if indexing is used to refer a this table element. See "Subscripting Using Index-Names (Indexing)" on page 53.

x A table without an INDEXED BY option can be referred to through indexing.

Indexes normally are allocated in static memory associated with the program containing the table. Thus, indexes are in the last-used state when a program is reentered.
x However, in the following cases, indexes are allocated on a per-invocation basis. Thus,
x you must SET the value of the index on every entry for indexes on tables in the:

x • Local-Storage Section

x • Working-Storage Section of a class definition (object instance variables)

x • Linkage Section of a:

x – Method
x – Program compiled with the RECURSIVE attribute
x – Program compiled with the THREAD option (Workstation Only)

**Note:** Indexes specified in an External data record do not possess the external attribute.

**index-name-1**

Must follow the rules for formation of user-defined words. At least one character must be alphabetic.

Each index-name specifies an index to be created by the compiler for use by the program. These index-names are **not** data-names, and are not identified elsewhere in the COBOL program; instead, they can be regarded as private special registers for the use of this object program only. They are not data, or part of any data hierarchy.

x As an IBM extension, unreferenced index names need not be uniquely defined.

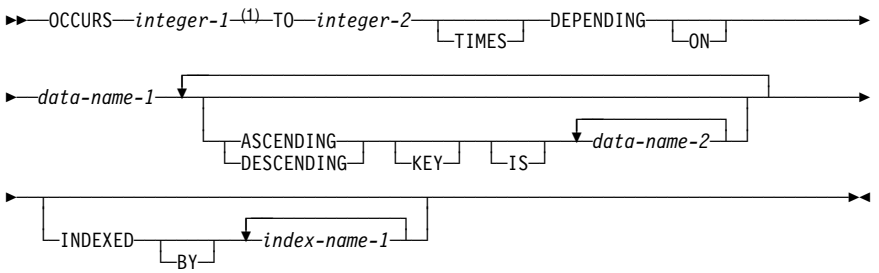
In one table entry, up to 12 index-names can be specified.

If a data item possessing the GLOBAL attribute includes a table accessed with an index, that index also possesses the GLOBAL attribute. Therefore, the scope of an index-name is identical to that of the data-name which names the table whose index is named by that index-name and the scope of name rules for data-names apply.

## Variable-Length Tables

Variable-length tables are specified using the OCCURS DEPENDING ON clause.

**Format 2—Variable-Length Tables**




**Note:**

x [1] Integer-1 is optional as an IBM extension. If integer-1 is omitted, a value of
x 1 is assumed and the key word TO must also be omitted.

**integer-1**

The minimum number of occurrences.

The value of integer-1 must be greater than or equal to zero; it must also be less than the value of integer-2.

**integer-2**

The maximum number of occurrences.

Integer-2 must be greater than integer-1.

The **length** of the subject item is fixed; it is only the **number of repetitions** of the subject item that is variable.

## OCCURS DEPENDING ON Clause

The OCCURS DEPENDING ON clause specifies variable-length tables.

**data-name-1**

Specifies the **object** of the OCCURS DEPENDING ON clause; that is, the data item whose current value represents the current number of occurrences of the subject item. The contents of items whose occurrence numbers exceed the value of the object are undefined.

Must describe an integer item.

The object of the OCCURS DEPENDING ON clause must not occupy any storage position within the range of the table (that is, any storage position from the first character position in the table through the last character position in the table).

x The object of the OCCURS DEPENDING ON clause may not be variably located;
x the object cannot follow an item that contains an OCCURS DEPENDING ON
x clause.

If the OCCURS clause is specified in a data description entry included in a record description entry containing the EXTERNAL clause, data-name-1, if specified, must reference a data item possessing the external attribute which is described in the same Data Division.

## OCCURS DEPENDING ON Clause

If the OCCURS clause is specified in a data description entry subordinate to one containing the GLOBAL clause, data-name-1, if specified, must be a global name and must reference a data item which is described in the same Data Division.

At the time that the group item, or any data item that contains a subordinate OCCURS DEPENDING ON item or that follows but is not subordinate to the OCCURS DEPENDING ON item, is referenced, the value of the object of the OCCURS DEPENDING ON clause must fall within the range integer-1 through integer-2.

When a group item containing a subordinate OCCURS DEPENDING ON item is referred to, the part of the table area used in the operation is determined as follows:

- If the object is outside the group, only that part of the table area that is specified by the object at the start of the operation will be used.
- If the object is included in the same group and the group data item is referenced as a sending item, only that part of the table area that is specified by the value of the object at the start of the operation will be used in the operation.
- If the object is included in the same group and the group data item is referenced as a receiving item, the maximum length of the group item will be used in the operation.

Following are the verbs that are affected by the maximum length rule:

- ACCEPT identifier (Format 1 and 2)
- CALL ... USING BY REFERENCE
- INVOKE ... USING BY REFERENCE
- MOVE ... TO identifier
- READ ... INTO identifier
- RELEASE identifier FROM ...
- RETURN ... INTO identifier
- REWRITE identifier FROM ...
- STRING ... INTO identifier
- UNSTRING ... INTO identifier DELIMITER IN identifier
- WRITE identifier FROM ...

The maximum length of variable-length groups is always used when they appear as the identifier on the CALL ... USING BY REFERENCE identifier statement. Therefore, the object of the OCCURS DEPENDING ON clause does not need to be set, unless the group is variably-located.

If the group item is followed by a non-subordinate item, the actual length, rather than the maximum length, will be used. At the time the subject of entry is referenced, or any data item subordinate or superordinate to the subject of entry is referenced, the object of the OCCURS DEPENDING ON clause must fall within the range integer-1 through integer-2.

In one record description entry, any entry that contains an OCCURS DEPENDING ON clause can be followed only by items subordinate to it.

The OCCURS DEPENDING ON clause cannot be specified as subordinate to another OCCURS clause.

The following constitute complex OCCURS DEPENDING ON:

- Subordinate items can contain OCCURS DEPENDING ON clauses.
- Entries containing an OCCURS DEPENDING ON clause can be followed by non-subordinate items. Non-subordinate items, however, cannot be the object of an OCCURS DEPENDING ON clause.
- The location of any subordinate or non-subordinate item, following an item containing an OCCURS DEPENDING ON clause, is affected by the value of the OCCURS DEPENDING ON object.
- Entries subordinate to the subject of an OCCURS DEPENDING ON clause can contain OCCURS DEPENDING ON clauses.
- When implicit redefinition is used in a File Description (FD) entry, subordinate level items can contain OCCURS DEPENDING ON clauses.
- The INDEXED BY phrase can be specified for a table that has a subordinate item that contains an OCCURS DEPENDING ON clause.

For more information on complex OCCURS DEPENDING ON, see the *IBM COBOL Programming Guide* for your platform.

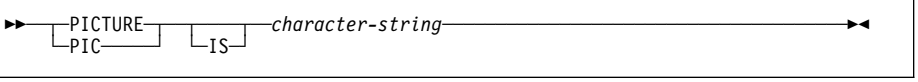
All data-names used in the OCCURS clause can be qualified; they can not be subscripted or indexed.

The ASCENDING/DESCENDING KEY and INDEXED BY clauses are described under "Fixed-Length Tables" on page 164.

## PICTURE Clause

The PICTURE clause specifies the general characteristics and editing requirements of an elementary item.

**Format**




**PICTURE or PIC**

The PICTURE clause must be specified for every elementary item except an index data item or the subject of the RENAMES clause. In these cases, use of this clause is prohibited.

The PICTURE clause can be specified only at the elementary level.

PIC is an abbreviation for PICTURE and has the same meaning.

## PICTURE Clause

**character-string**
PICTURE character-string is made up of certain COBOL characters used as symbols. The allowable combinations determine the category of the elementary data item.

The PICTURE character-string can contain a maximum of 30 characters.

The PICTURE clause is **not** allowed:

- For index data items or the subject of the RENAMES clause
- In descriptions of items described with USAGE IS INDEX
- For USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE data items
- For internal floating-point data items

## Symbols Used in the PICTURE Clause

The meaning of each PICTURE clause symbol is defined in Table 12 on page 171. The sequence in which PICTURE clause symbols must be specified is shown in Figure 6 on page 174. More detailed explanations of PICTURE clause symbols follow the figures.

Any punctuation character appearing within the PICTURE character-string is not considered a punctuation character, but rather a PICTURE character-string symbol.

When specified in the SPECIAL-NAMES paragraph, DECIMAL-POINT IS COMMA exchanges the functions of the period and the comma in PICTURE character strings and in numeric literals.

The lowercase letters corresponding to the uppercase letters representing the following PICTURE symbols are equivalent to their uppercase representations in a PICTURE character-string:

```
A, B, P, S, V, X, Z, CR, DB
E, G, N
```

The heading **Size** refers to the number of bytes the symbol contributes to the actual size of the data item.

*Table 12 (Page 1 of 3). PICTURE Clause Symbol Meanings*

| | Symbol | Meaning | Size | Restrictions |
|---|---|---|---|---|
| | A | A character position that can contain only a letter of the alphabet or a space. | Occupies 1 byte | |
| x | B | **For Non-DBCS data**—a character position into which the space character is inserted. | Occupies 1 byte | |
| x | | **For DBCS data**—a character position into which a DBCS space is inserted. Represents a single DBCS character position containing a DBCS space. | Occupies 2 bytes | |
| x | E | Marks the start of the exponent in an external floating-point item. | Occupies 1 byte | |
| x | G | A DBCS character position | Occupies 2 bytes | Cannot be specified for a non-DBCS item.<br>Under AIX, OS/2, and Windows, the locale you select must indicate a DBCS code page. For information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 543. |
| x | N | A DBCS character position | Occupies 2 bytes | Cannot be specified for a non-DBCS item.<br>Under AIX, OS/2, and Windows the locale you select must indicate a DBCS code page. For information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 543. |
| | P | An assumed decimal scaling position. Used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. See also "P Symbol" on page 175. | Not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric-edited items or in items that appear as arithmetic operands.<br>The size of the value is the number of digit positions represented by the PICTURE character-string. | Can appear only as a continuous string of Ps in the leftmost or rightmost digit positions within a PICTURE character-string. |

## PICTURE Clause

*Table 12 (Page 2 of 3). PICTURE Clause Symbol Meanings*

| Symbol | Meaning | Size | Restrictions |
|---|---|---|---|
| S | An indicator of the presence (but not the representation nor, necessarily, the position) of an operational sign. An operational sign indicates whether the value of an item involved in an operation is positive or negative. | Not counted in determining the size of the elementary item, unless an associated SIGN clause specifies the SEPARATE CHARACTER phrase (which would occupy 1 byte). | Must be written as the leftmost character in the PICTURE string. |
| V | An indicator of the location of the assumed decimal point. Does not represent a character position.<br>When the assumed decimal point is to the right of the rightmost symbol in the string, the V is redundant. | Not counted in the size of the elementary item | Can appear only once in a character-string. |
| X | A character position that can contain any allowable character from the character set of the computer. | Occupies 1 byte | |
| Z | A leading numeric character position. When that position contains a zero, a space character replaces the zero. | Each 'Z' is counted in the size of the data item. | |
| 9 | A character position that contains a numeral. | Each '9' is counted in the size of the data item. | |
| 0 | A character position into which the numeral zero is inserted. | Each '0' is counted in the size of the data item. | |
| / | A character position into which the slash character is inserted. | Each '/' is counted in the size of the data item. | |
| , | A character position into which a comma is inserted. | Each ',' is counted in the size of the data item. | If the comma insertion character is the last symbol in the PICTURE character-string, the PICTURE clause must be the last clause of the data description entry and must be immediately followed by the separator period.<br>A trailing comma insertion character can be immediately followed by the separator comma or separator semicolon; in this case, the PICTURE clause need not be the last clause of the data description entry. |

*Table 12 (Page 3 of 3). PICTURE Clause Symbol Meanings*

| Symbol | Meaning | Size | Restrictions |
|---|---|---|---|
| . | An editing symbol that represents the decimal point for alignment purposes. In addition, it represents a character position into which a period is inserted. | Each '.' is counted in the size of the data item. | If the period insertion character is the last symbol in the PICTURE character-string, the PICTURE clause must be the last clause of that data description entry and must be immediately followed by the separator period. |
| x x x x x x x | | | A trailing period insertion character can be immediately followed by the separator comma or separator semicolon; in this case, the PICTURE clause need not be the last clause of the data description entry. |
| + - CR DB | Editing sign control symbols. Each represents the character position into which the editing sign control symbol is placed. | Each character used in the symbol is counted in determining the size of the data item. | The symbols are mutually exclusive in one character-string. |
| * | A check protect symbol—a leading numeric character position into which an asterisk is placed when that position contains a zero. | Each asterisk (*) is counted in the size of the item. | |
| $ | A character position into which a currency symbol is placed. See also “Currency Symbol” on page 175. | The currency symbol is counted in the size of the data item. | |

## PICTURE Clause

Figure 6 shows the sequence in which PICTURE clause symbols must be specified.

| FIRST SYMBOL / SECOND SYMBOL | | Non-Floating Insertion Symbols | | | | | | | | | | Floating Insertion Symbols | | | | | | Other Symbols | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | B | 0 | / | , | . | + - | + - | CR DB | $ | E | Z * | Z * | + - | + - | $ | $ | 9 | A X | S | V | P | P | G | N |
| NON-FLOATING INSERTION SYMBOLS | B | ● | ● | ● | ● | ● | ● | | | ● | | ● | ● | ● | ● | ● | ● | ● | ● | | ● | | ● | ■ | |
| | 0 | ● | ● | ● | ● | ● | ● | | | ● | | ● | ● | ● | ● | ● | ● | ● | ● | | ● | | ● | | |
| | / | ● | ● | ● | ● | ● | ● | | | ● | | ● | ● | ● | ● | ● | ● | ● | ● | | ● | | ● | | |
| | , | ● | ● | ● | ● | ● | ● | | | ● | | ● | ● | ● | ● | ● | ● | ● | | | ● | | ● | | |
| | . | ● | ● | ● | ● | | ● | | | ● | | ● | | ● | | ● | | ● | | | | | | | |
| | + - | | | | | | | | | | | | | | | | | | | | | | | | |
| | + - | ● | ● | ● | ● | ● | | | | ● | ■ | ● | ● | | | ● | ● | ● | | | ● | ● | ● | | |
| | CR DB | ● | ● | ● | ● | ● | | | | ● | | ● | ● | | | ● | ● | ● | | | ● | ● | ● | | |
| | $ | | | | | | ● | | | | | | | | | | | | | | | | | | |
| | E | | | | ■ | ■ | | | | | | | | | | | | ■ | | | ■ | | | | |
| FLOATING INSERTION SYMBOLS | Z * | ● | ● | ● | ● | | ● | | | ● | | ● | | | | | | | | | | | | | |
| | Z * | ● | ● | ● | ● | ● | ● | | | ● | | ● | ● | | | | | | | | ● | | ● | | |
| | + - | ● | ● | ● | ● | | | | | ● | | | | ● | | | | | | | | | | | |
| | + - | ● | ● | ● | ● | ● | | | | ● | | | | | ● | ● | | | | | ● | | ● | | |
| | $ | ● | ● | ● | ● | | ● | | | | | | | | | ● | | | | | | | | | |
| | $ | ● | ● | ● | ● | ● | ● | | | | | | | | ● | ● | | | | | ● | | ● | | |
| OTHER SYMBOLS | 9 | ● | ● | ● | ● | ● | ● | | | ● | | ● | | ● | | ● | | ● | ● | ● | ● | | ● | | |
| | A X | ● | ● | ● | | | | | | | | | | | | | | ● | ● | | | | | | |
| | S | | | | | | | | | | | | | | | | | | | | | | | | |
| | V | ● | ● | ● | ● | | ● | | | ● | | ● | | ● | | ● | | ● | | ● | | ● | | | |
| | P | ● | ● | ● | ● | | ● | | | ● | | ● | | ● | | ● | | ● | | ● | | ● | | | |
| | P | | | | | | ● | | | ● | | | | | | | | | | ● | ● | | ● | | |
| | G | ■ | | | | | | | | | | | | | | | | | | | | | | ■ | |
| | N | | | | | | | | | | | | | | | | | | | | | | | | ■ |

*Figure 6. PICTURE Clause Symbol Sequence*

**Figure Legend:**

| | |
|---|---|
| • | Closed circle indicates that the symbol(s) at the top of the column can, in a given character-string, appear anywhere to the left of the symbol(s) at the left of the row. |
| x ■ x | Closed square indicates that the item is an IBM extension. |
| **{ }** | Braces indicate items that are mutually exclusive. |
| **Symbols that appear twice** | Nonfloating insertion symbols + and -, floating insertion symbols Z, *, +, -, and $, and the symbol P appear twice. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the table represents its use to the right of the decimal point position. |

## P Symbol

Because the scaling position character P implies an assumed decimal point (to the left of the Ps, if the Ps are leftmost PICTURE characters; to the right of the Ps, if the Ps are rightmost PICTURE characters), the assumed decimal point symbol, V, is redundant as either the leftmost or rightmost character within such a PICTURE description.

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 character representation of the data item. This algebraic value assumes the decimal point in the prescribed location and zero in place of the digit position specified by the symbol P. The size of the value is the number of digit positions represented by the PICTURE character-string. These operations are any of the following:

- Any operation requiring a numeric sending operand.
- A MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol P.
- A MOVE statement where the sending operand is numeric-edited and its PICTURE character-string contains the symbol P and the receiving operand is numeric or numeric-edited.
- 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.

## Currency Symbol

The currency symbol in a character-string is represented either by the symbol $ or by x the single character specified in the CURRENCY compiler option or in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division.

If the NOCURRENCY compiler option is in effect, the $ character is the default value for the currency symbol.

## PICTURE Clause

### Character-String Representation

#### Symbols That Can Appear More Than Once

The following symbols can appear more than once in one PICTURE character-string:

```
| A  B  P  X  Z  9  0  /  ,  +  -  *  $
x G  N
```

At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, -, or $ must be present in a PICTURE string.

x The symbol G or N can appear alone in the PICTURE character-string.

An unsigned nonzero integer enclosed in parentheses immediately following any of these symbols specifies the number of consecutive occurrences of that symbol.

**Example:** The following two PICTURE clause specifications are equivalent:

```
PICTURE IS $99999.99CR

PICTURE IS $9(5).9(2)CR
```

#### Symbols That Can Appear Only Once

The following symbols can appear only once in one PICTURE character-string:

```
S  V  .  CR  DB
x E
```

Except for the PICTURE symbol V, each time any of the above symbols appears in the character-string, it represents an occurrence of that character or set of allowable characters in the data item.

### Data Categories and PICTURE Rules

The allowable combinations of PICTURE symbols determine the data category of the item.

- Alphabetic items
- Numeric Items
- Numeric-edited items
- Alphanumeric items
- Alphanumeric-edited items
- x DBCS items
- x External floating-point items

### Alphabetic Items

The PICTURE character-string can contain only the symbol A.

The contents of the item in standard data format must consist of any of the letters of the English alphabet and the space character.

***Other Clauses:*** USAGE DISPLAY must be specified or implied.

Any associated VALUE clause must specify a nonnumeric literal containing only alphabetic characters, SPACE, or a symbolic-character as the value of a figurative constant.

## Numeric Items

Types of numeric items are:

- Binary
- Packed decimal (internal decimal)
- Zoned decimal (external decimal)

The PICTURE character-string can contain only the symbols 9, P, S, and V.

The number of digit positions must range from 1 through 18, inclusive.

If unsigned, the contents of the item in standard data format must contain a combination of the Arabic numerals 0-9. If signed, it may also contain a +, –, or other representation of the operational sign.

***Examples of Valid Ranges***

```
   PICTURE   Valid Range of Values

     9999     0 through 9999
      S99    -99 through +99
   S999V9    -999.9 through +999.9
   PPP999     0 through .000999
  S999PPP    -1000 through -999000 and
             +1000 through +999000 or zero
```

***Other Clauses:*** The USAGE of the item can be DISPLAY, BINARY, COMPUTATIONAL, PACKED-DECIMAL, COMPUTATIONAL-3, COMPUTATIONAL-4, or COMPUTATIONAL-5.

A VALUE clause can specify a figurative constant ZERO.

A VALUE clause associated with an elementary numeric item must specify a numeric literal or the figurative constant ZERO. A VALUE clause associated with a group item consisting of elementary numeric items must specify a **nonnumeric** literal or a figurative constant, because the group is considered alphanumeric. In both cases, the literal is treated exactly as specified; no editing is performed.

The NUMPROC, TRUNC, and BINARY compiler options can affect the use of numeric data items. For details, see the *IBM COBOL Programming Guide* for your platform. (Note, BINARY is only available on AIX, OS/2, and Windows.)

## Numeric-edited Items

The PICTURE character-string can contain the following symbols:

```
B  P  V  Z  9  0  /  ,  .  +  -  CR  DB  *  $
```

The combinations of symbols allowed are determined from the PICTURE clause symbol order allowed (see Figure 6 on page 174), and the editing rules (see "PICTURE Clause Editing" on page 180).

The following rules also apply:

- Either the BLANK WHEN ZERO clause must be specified for the item, or the string must contain at least one of the following symbols:

  ```
  B  /  Z  0  ,  .  *  +  -  CR  DB  $
  ```

- The number of digit positions represented in the character-string must be in the range 1 through 18, inclusive.
- The total number of character positions in the string (including editing-character positions) must not exceed 249.

The contents of those character positions representing digits in standard data format must be one of the 10 Arabic numerals.

***Other Clauses:*** USAGE DISPLAY must be specified or implied.

Any associated VALUE clause must specify a nonnumeric literal or a figurative constant. The literal is treated exactly as specified; no editing is done.

### Alphanumeric Items

The PICTURE character-string must consist of either of the following:

- The symbol X
- Combinations of the symbols A, X, and 9 (A character-string containing all As or all 9s does not define an alphanumeric item.)

The item is treated as if the character-string contained only the symbol X.

The contents of the item in standard data format may be any allowable characters from the character set of the computer.

***Other Clauses:*** USAGE DISPLAY must be specified or implied.

Any associated VALUE clause must specify a nonnumeric literal or a figurative constant.

### Alphanumeric-edited Items

The PICTURE character-string can contain the following symbols:

```
A  X  9  B  0  /
```

The string must contain at least one A or X, and at least one B or 0 (zero) or /.

The contents of the item in standard data format must be two or more characters from the character set of the computer.

***Other Clauses:*** USAGE DISPLAY must be specified or implied.

Any associated VALUE clause must specify a nonnumeric literal or a figurative constant. The literal is treated exactly as specified; no editing is done.

## DBCS Items

The PICTURE character-string can contain the symbol(s) G, G and B, or N. Each G, B or N represents a single DBCS character position.

The entire range of characters for a DBCS literal can be used.

WRKSTN Under AIX, OS/2, and Windows, do not include a single byte character of a DBCS code page in a DBCS data item. (The locale you select must indicate a DBCS code page. For information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 543.)

For a code page with characters represented in double bytes, the following padding and truncation rules apply:

- Padding—For DBCS data items, padding is done using the double byte space characters until the data area is filled (based on the number of byte positions allocated for the data item).

  Single-byte characters are used for padding when the padding needed is not a multiple of the code page width (for example, a group item moved to a DBCS data item).
- Truncation—For DBCS data items, truncation is done based on the size of the target data area on the byte boundary of the end of the data area. You must ensure that a truncation does not result in truncation of bytes representing a partial DBCS character.

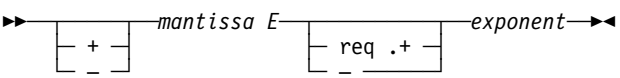
***Other Clauses:*** When PICTURE clause symbol G is used, USAGE DISPLAY-1 must be specified.

When PICTURE clause symbol N is used, USAGE DISPLAY-1 is assumed and does not need to be specified.

Any associated VALUE clause must specify a DBCS literal or the figurative constant SPACE/SPACES.

## External Floating-point Items

**Format**




**+ or –**

A sign character must immediately precede both the mantissa and the exponent.

A + sign indicates that a positive sign will be used in the output to represent positive values and that a negative sign will represent negative values.

A – sign indicates that a blank will be used in the output to represent positive values and that a negative sign will represent negative values.

## PICTURE Clause

x Each sign position occupies one byte of storage.

x **mantissa**
x The mantissa can contain the symbols:

x `9 . V`

x An actual decimal point can be represented with a period (.) while an assumed
x decimal point is represented by a V.

x Either an actual or an assumed decimal point must be present in the mantissa; the
x decimal point can be leading, embedded, or trailing.

x The mantissa can contain from 1 to 16 numeric characters.

x **E** Indicates the exponent.

x **exponent**
x The exponent must consist of the symbol 99.

| ***Other Clauses:*** The OCCURS, REDEFINES, RENAMES, and USAGE clauses may
x be associated with external floating-point items.

x The SIGN clause is accepted as documentation and has no effect on the representation
x of the sign.

x The SYNCHRONIZED clause is treated as documentation.

x The following clauses are invalid with external floating-point items:

x • BLANK WHEN ZERO
x • JUSTIFIED
x • VALUE

## PICTURE Clause Editing

There are two general methods of editing in a PICTURE clause:

- Insertion editing
  - Simple insertion
  - Special insertion
  - Fixed insertion
  - Floating insertion
- Suppression and replacement editing
  - Zero suppression and replacement with asterisks
  - Zero suppression and replacement with spaces.

The type of editing allowed for an item depends on its **data category**. The type of editing that is valid for each category is shown

*Table 13. Data Categories*

| Data Category | Type of Editing | Insertion Symbol |
|---|---|---|
| Alphabetic | None | None |
| Numeric | None | None |
| Numeric-edited | Simple insertion | B 0 / , |
| | Special insertion | . |
| | Fixed insertion | $ + – CR DB |
| | Floating insertion | $ + – |
| | Zero suppression | Z * |
| | Replacement | Z * + – $ |
| Alphanumeric | None | None |
| Alphanumeric-edited | Simple insertion | B 0 / |
| x DBCS | Simple insertion | B |
| x External floating-point | Special insertion | . |

## Simple Insertion Editing

x This type of editing is valid for alphanumeric-edited, numeric-edited, and DBCS items.

Each insertion symbol is counted in the size of the item, and represents the position
x within the item where the equivalent character is to be inserted. For edited DBCS
x items, each insertion symbol (B) is counted in the size of the item and represents the
x position within the item where the DBCS space is to be inserted.

For example:

```
PICTURE          Value of Data        Edited Result

X(10)/XX         ALPHANUMER01         ALPHANUMER/01
X(5)BX(7)        ALPHANUMERIC         ALPHA NUMERIC
99,B999,B000     1234                 01,b234,b000
99,999           12345                12,345
x GGBBGG         D1D2D3D4             D1D2bbbbD3D4
```

## Special Insertion Editing

x This type of editing is valid for either numeric-edited items or external floating-point
x items.

The period (.) is the special insertion symbol; it also represents the actual decimal point for alignment purposes.

The period insertion symbol is counted in the size of the item, and represents the position within the item where the actual decimal point is inserted.

Either the actual decimal point or the symbol V as the assumed decimal point, but not both, must be specified in one PICTURE character-string.

For example:

| PICTURE | Value of Data | Edited Results |
|---|---|---|
| 999.99 | 1.234 | 001.23 |
| 999.99 | 12.34 | 012.34 |
| 999.99 | 123.45 | 123.45 |
| 999.99 | 1234.5 | 234.50 |
| x +999.99E+99 | 12345 | +123.45E+02 |

## Fixed Insertion Editing

This type of editing is valid only for numeric-edited items. The following insertion symbols are used:

$ (or other valid currency symbol)

+ – CR DB (editing-sign control symbols)

In fixed insertion editing, only one currency symbol and one editing sign control symbol can be specified in one PICTURE character-string.

Unless it is preceded by a + or – symbol, the currency symbol must be the first character in the character-string.

When either + or – is used as a symbol, it must be the first or last character in the character-string.

When CR or DB is used as a symbol, it must occupy the rightmost two character positions in the character-string. If these two character positions contain the symbols CR or DB, the uppercase letters are the insertion characters.

Editing sign control symbols produce results that depend on the value of the data item, as shown below:

| Editing Symbol in PICTURE Character-String | Result: Data Item Positive or Zero | Result: Data Item Negative |
|---|---|---|
| + | + | - |
| - | space | - |
| CR | 2 spaces | CR |
| DB | 2 spaces | DB |

For example:

```
PICTURE         Value of Data    Edited Result

 999.99+         +6555.556         555.55+
+9999.99         -6555.555        -6555.55
 9999.99         +1234.56          1234.56
 $999.99          -123.45          $123.45
-$999.99          -123.456        -$123.45
-$999.99          +123.456         $123.45
$9999.99CR        +123.45         $0123.45
$9999.99DB        -123.45         $0123.45DB
```

## Floating Insertion Editing

This type of editing is valid only for numeric-edited items.

The following symbols are used:

```
$ + -
```

Within one PICTURE character-string, these symbols are mutually exclusive as floating insertion characters.

Floating insertion editing is specified by using a string of at least two of the allowable floating insertion symbols to represent leftmost character positions into which these actual characters can be inserted.

The leftmost floating insertion symbol in the character-string represents the leftmost limit at which this actual character can appear in the data item. The rightmost floating insertion symbol represents the rightmost limit at which this actual character can appear.

The second leftmost floating insertion symbol in the character-string represents the left-most limit at which numeric data can appear within the data item. Nonzero numeric data may replace all characters at or to the right of this limit.

Any simple-insertion symbols (B 0 / ,) within or to the immediate right of the string of floating insertion symbols are considered part of the floating character-string. If the period (.) special-insertion symbol is included within the floating string, it is considered to be part of the character-string.

To avoid truncation, the minimum size of the PICTURE character-string must be:

- The number of character positions in the sending item, plus
- The number of nonfloating insertion symbols in the receiving item, plus
- One character for the floating insertion symbol.

### Representing Floating Insertion Editing

In a PICTURE character-string, there are two ways to represent floating insertion editing and, thus, two ways in which editing is performed:

1. Any or all leading numeric character positions to the left of the decimal point are represented by the floating insertion symbol. When editing is performed, a single floating insertion character is placed to the immediate left of the first nonzero digit in the data, or of the decimal point, whichever is farther to the left. The character positions to the left of the inserted character are filled with spaces.

   If all numeric character positions in the PICTURE character-string are represented by the insertion character, then at least one of the insertion characters must be to the left of the decimal point.

2. All the numeric character positions are represented by the floating insertion symbol. When editing is performed, then:

   - If the value of the data is zero, the entire data item will contain spaces.
   - If the value of the data is nonzero, the result is the same as in rule 1.

For example:

| PICTURE | Value of Data | Edited Result |
|---|---|---|
| $$$$.99 | .123 | $.12 |
| $$$9.99 | .12 | $0.12 |
| $,$$$,999.99 | -1234.56 | $1,234.56 |
| +,+++,999.99 | -123456.789 | -123,456.78 |
| $$,$$$,$$$.99CR | -1234567 | $1,234,567.00CR |
| ++,+++,+++.+++ | 0000.00 | |

## Zero Suppression and Replacement Editing

This type of editing is valid only for numeric-edited items.

In zero suppression editing, the symbols Z and * are used. These symbols are mutually exclusive in one PICTURE character-string.

The following symbols are mutually exclusive as floating replacement symbols in one PICTURE character-string:

Z * + – $

Specify zero suppression and replacement editing with a string of one or more of the allowable symbols to represent leftmost character positions in which zero suppression and replacement editing can be performed.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating editing symbols are considered part of the string. If the period (.) special insertion symbol is included within the floating editing string, it is considered to be part of the character-string.

## Representing Zero Suppression

In a PICTURE character-string, there are two ways to represent zero suppression, and two ways in which editing is performed:

1. Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols. When editing is performed, the replacement character replaces any leading zero in the data that appears in the same character position as a suppression symbol. Suppression stops at the leftmost character:
   - That does not correspond to a suppression symbol
   - That contains nonzero data
   - That is the decimal point.
2. All the numeric character positions in the PICTURE character-string are represented by the suppression symbols. When editing is performed, and the value of the data is nonzero, the result is the same as in the preceding rule. If the value of the data is zero, then:
   - If Z has been specified, the entire data item will contain spaces.
   - If * has been specified, the entire data item, except the actual decimal point, will contain asterisks.

For example:

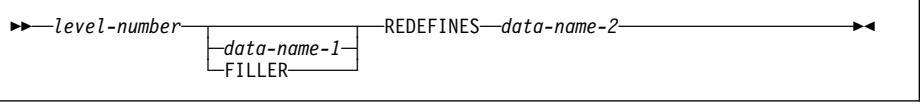
| PICTURE | Value of Data | Edited Result |
|---|---|---|
| ****.** | 0000.00 | ****.** |
| ZZZZ.ZZ | 0000.00 | |
| ZZZZ.99 | 0000.00 | .00 |
| ****.99 | 0000.00 | ****.00 |
| ZZ99.99 | 0000.00 | 00.00 |
| Z,ZZZ.ZZ+ | +123.456 | 123.45+ |
| *,***.**+ | -123.45 | **123.45- |
| **,***,***.**+ | +12345678.9 | 12,345,678.90+ |
| $Z,ZZZ,ZZZ.ZZCR | +12345.67 | $ 12,345.67 |
| $B*,***,***.**BBDB | -12345.67 | $ ***12,345.67 DB |

Do not specify both the asterisk (*) as a suppression symbol and the BLANK WHEN ZERO clause for the same entry.

## REDEFINES Clause

The REDEFINES clause allows you to use different data description entries to describe the same computer storage area.

**Format**




**Note:** Level-number, data-name-1, and FILLER are not part of the REDEFINES clause itself, and are included in the format only for clarity.

When specified, the REDEFINES clause must be the first entry following data-name-1 or FILLER. If data-name-1 or FILLER is not specified, the REDEFINES clause must be the first entry following the level-number.

The level-numbers of data-name-1 and data-name-2 must be identical, and must not be level 66 or level 88.

**data-name-1, FILLER**
Identifies an alternate description for the same area, and is the **redefining** item or the **REDEFINES subject**.

**data-name-2**
Is the **redefined** item or the **REDEFINES object**.

When more that one level-01 entry is written subordinate to an FD entry, a condition known as implicit redefinition occurs. That is, the second level-01 entry implicitly redefines the storage allotted for the first entry. In such level-01 entries, the REDEFINES clause must not be specified.

Redefinition begins at data-name-1 and ends when a level-number less than or equal to that of data-name-1 is encountered. No entry having a level-number numerically lower than those of data-name-1 and data-name-2 can occur between these entries. For example:

```
05    A PICTURE X(6).
05    B REDEFINES A.
  10 B-1                    PICTURE X(2).
  10 B-2                    PICTURE 9(4).
05   C                      PICTURE 99V99.
```

In this example, A is the redefined item, and B is the redefining item. Redefinition begins with B and includes the two subordinate items B-1 and B-2. Redefinition ends when the level-05 item C is encountered.

x The data description entry for data-name-2, the redefined item, can contain a REDE-
x FINES clause.

The data description entry for the redefined item cannot contain an OCCURS clause. However, the redefined item can be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to the redefined item in the REDEFINES clause must not be subscripted. Neither the redefined item nor the redefining item, or any items subordinate to them, can contain an OCCURS DEPENDING ON clause.

If the GLOBAL clause is used in the data description entry which contains the REDEFINES clause, it is only the subject of that REDEFINES clause that possesses the global attribute.

The EXTERNAL clause must not be specified on the same data description entry as a REDEFINES clause.

If the data item referenced by data-name-2 is either declared to be an external data record or is specified with a level-number other than 01, the number of character positions it contains must be greater than or equal to the number of character positions in the data item referenced by the subject of this entry. If the data-name referenced by data-name-2 is specified with a level-number of 01 and is not declared to be an external data record, there is no such constraint.

x When the data item implicitly redefines multiple 01-level records in a file description
x (FD) entry, items subordinate to the redefining or redefined item can contain an
x OCCURS DEPENDING ON clause.

One or more redefinitions of the same storage area are permitted. The entries giving the new descriptions of the storage area must immediately follow the description of the redefined area without intervening entries that define new character positions. Multiple redefinitions must all use the data-name of the original entry that defined this storage area. For example:

```
05    A                       PICTURE 9999.
05    B REDEFINES A           PICTURE 9V999.
05    C REDEFINES A           PICTURE 99V99.
```

The redefining entry (identified by data-name-1), and any subordinate entries, must not contain any VALUE clauses.

x An item described as USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or
x USAGE IS OBJECT REFERENCE can be the subject or object of a REDEFINES
x clause.

x An external or internal floating-point item can be the subject or object of a REDEFINES
x clause.

## REDEFINES Clause Considerations

Data items within an area can be redefined without changing their lengths. For example:

```
05  NAME-2.
  10  SALARY                  PICTURE XXX.
  10  SO-SEC-NO               PICTURE X(9).
  10  MONTH                   PICTURE XX.
05  NAME-1 REDEFINES NAME-2.
  10  WAGE                    PICTURE XXX.
  10  EMP-NO                  PICTURE X(9).
  10  YEAR                    PICTURE XX.
```

Data item lengths and types can also be re-specified within an area. For example:

```
05  NAME-2.
  10  SALARY                  PICTURE XXX.
  10  SO-SEC-NO               PICTURE X(9).
  10  MONTH                   PICTURE XX.
05  NAME-1 REDEFINES NAME-2.
  10  WAGE                    PICTURE 999V999.
  10  EMP-NO                  PICTURE X(6).
  10  YEAR                    PICTURE XX.
```

When an area is redefined, all descriptions of the area are always in effect; that is, redefinition does not cause any data to be erased and never supersedes a previous description. Thus, if B REDEFINES C has been specified, either of the two procedural statements, `MOVE X TO B` and `MOVE Y TO C`, could be executed at any point in the program.

In the first case, the area described as B would assume the value and format of X. In the second case, the same physical area (described now as C) would assume the value and format of Y. Note that, if the second statement is executed immediately after the first, the value of Y replaces the value of X in the one storage area.

The usage of a redefining data item need not be the same as that of a redefined item. This does not, however, cause any change in existing data. For example:

```
05  B                     PICTURE  99 USAGE DISPLAY VALUE 8.
05  C REDEFINES B         PICTURE S99 USAGE COMPUTATIONAL-4.
05  A                     PICTURE S99 USAGE COMPUTATIONAL-4.
```

Redefining B does not change the bit configuration of the data in the storage area. Therefore, the following two statements produce different results:

```
ADD B TO A
ADD C TO A
```

In the first case, the value 8 is added to A (because B has `USAGE DISPLAY`). In the second statement, the value -3848 is added to A (because C has `USAGE COMPUTATIONAL-4`), and the bit configuration of the storage area has the binary value -3848.

The above example demonstrates how the improper use of redefinition can give unexpected or incorrect results.

## REDEFINES Clause Examples

The REDEFINES clause can be specified for an item within the scope of an area being redefined (that is, an item subordinate to a redefined item). For example:

```
05  REGULAR-EMPLOYEE.
  10  LOCATION                      PICTURE A(8).
  10  GRADE                         PICTURE X(4).
  10  SEMI-MONTHLY-PAY              PICTURE 9999V99.
  10  WEEKLY-PAY REDEFINES SEMI-MONTHLY-PAY
                                    PICTURE 999V999.

05  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
  10  LOCATION                      PICTURE A(8).
  10  FILLER                        PICTURE X(6).
  10  HOURLY-PAY                    PICTURE 99V99.
```

The REDEFINES clause can also be specified for an item subordinate to a redefining item. For example:

```
05  REGULAR-EMPLOYEE.
  10  LOCATION                      PICTURE A(8).
  10  GRADE                         PICTURE X(4).
  10  SEMI-MONTHLY-PAY              PICTURE 999V999.

05  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
  10  LOCATION                      PICTURE A(8).
  10  FILLER                        PICTURE X(6).
  10  HOURLY-PAY                    PICTURE 99V99.
  10  CODE-H REDEFINES HOURLY-PAY    PICTURE 9999.
```

## Undefined Results

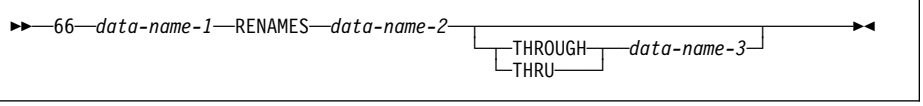
Undefined results can occur when:

- A redefining item is moved to a redefined item (that is, if `B REDEFINES C` and the statement `MOVE B TO C` is executed)
- A redefined item is moved to a redefining item (that is, if `B REDEFINES C` and if the statement `MOVE C TO B` is executed).

## RENAMES Clause

The RENAMES clause specifies alternative, possibly overlapping, groupings of elementary data items.

**Format**




The special level-number 66 must be specified for data description entries that contain the RENAMES clause. Level-number 66 and data-name-1 are not part of the RENAMES clause itself, and are included in the format only for clarity.

One or more RENAMES entries can be written for a logical record. All RENAMES entries associated with one logical record must immediately follow that record's last data description entry.

**data-name-1**
Identifies an alternative grouping of data items.

A level-66 entry cannot rename a level-01, level-77, level-88, or another level-66 entry.

Data-name-1 cannot be used as a qualifier; it can be qualified only by the names of level indicator entries or level-01 entries.

x Can specify a DBCS data item if data-name-2 specifies a DBCS data item and the
x THROUGH phrase is not specified.

**data-name-2, data-name-3**
Identify the original grouping of elementary data items; that is, they must name elementary or group items within the associated level-01 entry, and must not be the same data-name. Both data-names can be qualified.

The OCCURS clause must not be specified in the data entries for data-name-2 and data-name-3, or for any group entry to which they are subordinate. In addition, the OCCURS DEPENDING ON clause must not be specified for any item defined between data-name-2 and data-name-3.

When data-name-3 is specified, data-name-1 is treated as a group item that includes all elementary items:

- Starting with data-name-2 (if it is an elementary item) or the first elementary item within data-name-2 (if it is a group item)
- Ending with data-name-3 (if it is an elementary item) or the last elementary item within data-name-3 (if it is a group item).

The key words THROUGH and THRU are equivalent.

The leftmost character in data-name-3 must not precede the leftmost character in data-name-2; the rightmost character in data-name-3 must not precede the right-

most character in data-name-2. This means that data-name-3 cannot be totally subordinate to data-name-2.

When data-name-3 is not specified, all of the data attributes of data-name-2 become the data attributes for data-name-1. That is:

- When data-name-2 is a group item, data-name-1 is treated as a group item.
- When data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

Figure 7 illustrates valid and invalid RENAMES clause specifications.

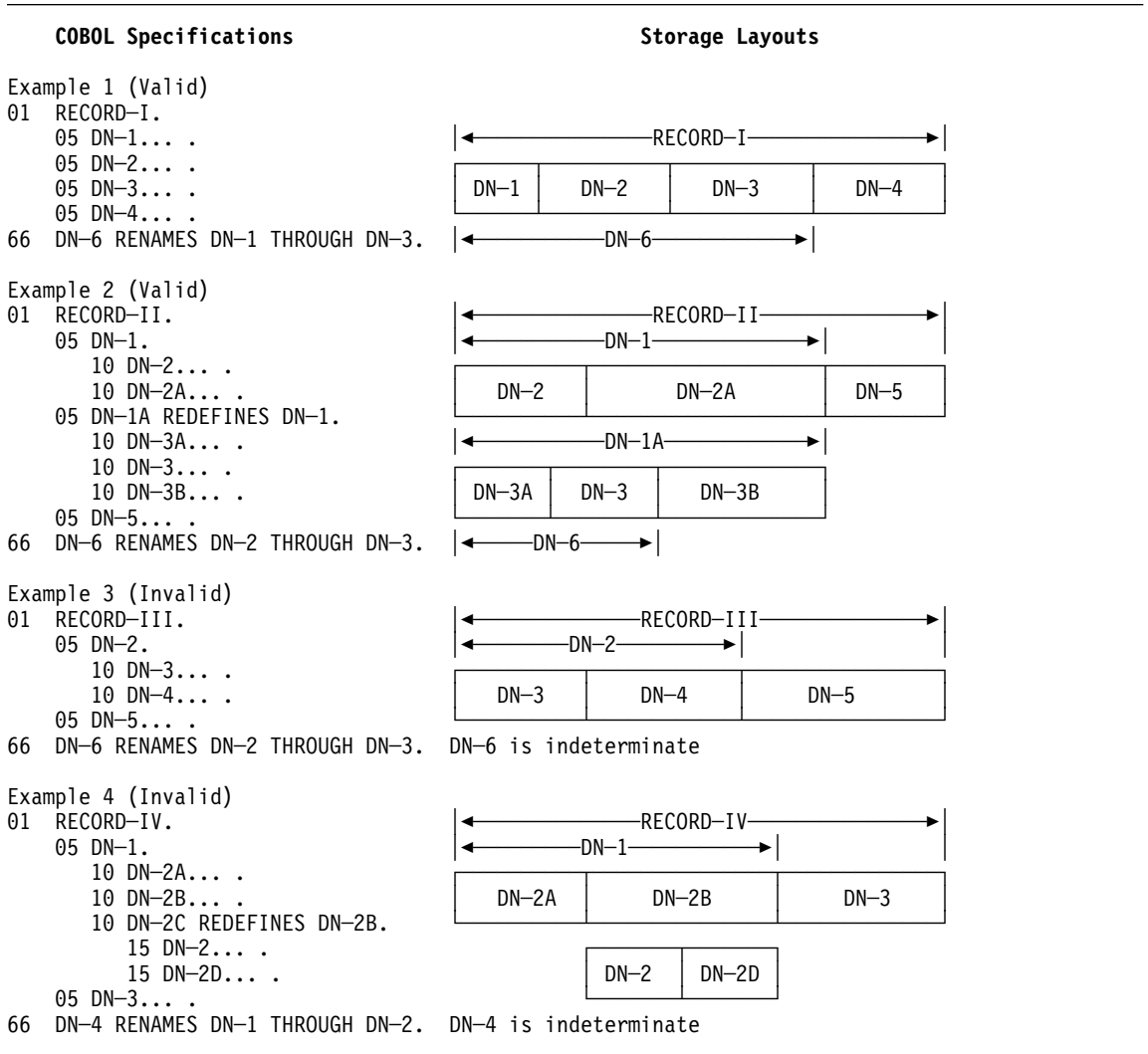
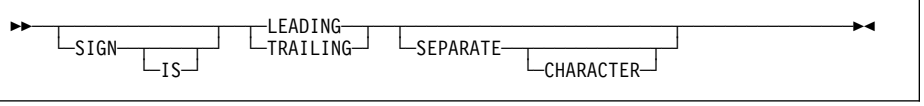



*Figure 7. RENAMES Clause—Valid and Invalid Specifications*

## SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign for a numeric entry.

**Format**




The SIGN clause can be specified only for a signed numeric data description entry (that is, one whose PICTURE character-string contains an S), or for a group item that contains at least one such elementary entry. USAGE IS DISPLAY must be specified, explicitly or implicitly.

If a SIGN clause is specified in either an elementary or group entry subordinate to a group item for which a SIGN clause is specified, the SIGN clause for the subordinate entry takes precedence for the subordinate entry.

If you specify the CODE-SET clause in an FD entry, any signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause.

The SIGN clause is required only when an explicit description of the properties and/or position of the operational sign is necessary.

When specified, the SIGN clause defines the position and mode of representation of the operational sign for the numeric data description entry to which it applies, or for each signed numeric data description entry subordinate to the group to which it applies.

If the SEPARATE CHARACTER phrase is not specified, then:

- The operational sign is presumed to be associated with the LEADING or TRAILING digit position, whichever is specified, of the elementary numeric data item. (In this instance, specification of SIGN IS TRAILING is the equivalent of the standard action of the compiler.)
- The character S in the PICTURE character string is not counted in determining the size of the item (in terms of standard data format characters).
- If the SEPARATE CHARACTER phrase is specified, then:
- The operational sign is presumed to be the LEADING or TRAILING character position, whichever is specified, of the elementary numeric data item. This character position is not a digit position.
- The character S in the PICTURE character string is counted in determining the size of the data item (in terms of standard data format characters).
- + is the character used for the positive operational sign.
- - is the character used for the negative operational sign.

Every numeric data description entry whose PICTURE contains the symbol S is a signed numeric data description entry. If the SIGN clause is also specified for such an entry, and conversion is necessary for computations or comparisons, the conversion takes place automatically.

x The SIGN clause is treated as documentation for external floating-point items. For
x internal floating-point items, the SIGN clause is invalid and will result in an S-level diag-
x nostic message.

## SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on a natural boundary in storage.

**Format**

SYNC is an abbreviation for SYNCHRONIZED and has the same meaning.

The SYNCHRONIZED clause is never required, but can improve performance on some systems for binary items used in arithmetic.

x The SYNCHRONIZED clause can appear at the elementary level or at the group level
x in which case, every elementary item within this group level item is synchronized.

**LEFT**
Specifies that the elementary item is to be positioned so that it will begin at the left character position of the natural boundary in which the elementary item is placed.

**RIGHT**
Specifies that the elementary item is to be positioned such that it will terminate on the right character position of the natural boundary in which it has been placed.

When specified, the LEFT and the RIGHT phrases are syntax checked, but they have no effect on the execution of the program.

The length of an elementary item is not affected by the SYNCHRONIZED clause.

## SYNCHRONIZED Clause

Table 14 lists the affect of the SYNCHRONIZE clause on other language elements.

*Table 14. SYNCHRONIZE Clause Affect on Other Language Elements*

| Language Element | Comments |
|---|---|
| OCCURS clause | When specified for an item within the scope of an OCCURS clause, each occurrence of the item is synchronized. |
| DISPLAY or PACKED-DECIMAL | Each item is syntax checked, but it has no effect on the execution of the program. |
| BINARY or COMPUTATIONAL | When the item is the first elementary item subordinate to an item that contains a REDEFINES clause, the item must not require the addition of unused character positions.<br><br>When the synchronized clause is not specified for a subordinate data item (one with a level number of 02 through 49):<br><br>• The item is aligned at a displacement that is a multiple of 2 relative to the beginning of the record, if its USAGE is BINARY and its PICTURE is in the range of S9 through S9(4).<br><br>• The item is aligned at a displacement that is a multiple of 4 relative to the beginning of the record, if its USAGE is BINARY and its PICTURE is in the range of S9(5) through S9(18), or its USAGE is INDEX.<br><br>When SYNCHRONIZED is not specified for binary items, no space is reserved for slack bytes. |
| USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE | The data is aligned on a fullword boundary. |
| COMPUTATIONAL-1 | The data is aligned on a fullword boundary. |
| COMPUTATIONAL-2 | The data is aligned on a doubleword boundary. |
| COMPUTATIONAL-3 | The data is treated the same as the SYNCHRONIZED clause for a PACKED-DECIMAL item. |
| COMPUTATIONAL-4 | The data is treated the same as the SYNCHRONIZED clause for a COMPUTATIONAL item. |
| COMPUTATIONAL-5 (Workstation Only) | The data is treated the same as the SYNCHRONIZED clause for a COMPUTATIONAL item. |
| DBCS and Floating Point Item | The SYNCHRONIZED clause is ignored. |
| REDEFINES clause | For an item that contains a REDEFINES clause, the data item that is redefined must have the proper boundary alignment for the data item that redefines it. For example, if you write the following, be sure that data item A begins on a fullword boundary:<br><br>`02 A PICTURE X(4).`<br>`02 B REDEFINES A PICTURE S9(9) BINARY SYNC.` |

In the File Section, the compiler assumes that all level-01 records containing SYNCHRONIZED items are aligned on doubleword boundaries in the buffer. You must provide the necessary slack bytes between records to ensure alignment when there are multiple records in a block.

In the Working-Storage Section, the compiler aligns all level-01 entries on a doubleword boundary.

For the purposes of aligning binary items in the Linkage Section, all level-01 items are assumed to begin on doubleword boundaries. Therefore, if you issue a CALL statement, such operands of any USING phrase within it must be aligned correspondingly.

## Slack Bytes

There are two types of slack bytes:

**Slack bytes *within* records**
Unused character positions preceding each synchronized item in the record.

**Slack bytes *between* records**
Unused character positions added between blocked logical records.

## Slack Bytes within Records

For any data description that has binary items that are not on their natural boundaries, the compiler inserts slack bytes within a record to ensure that all SYNCHRONIZED items are on their proper boundaries.

Because it is important that you know the length of the records in a file, you need to determine whether slack bytes are required and, if necessary, how many the compiler will add. The algorithm the compiler uses to calculate this is as follows:

- The total number of bytes occupied by all elementary data items preceding the binary item are added together, including any slack bytes previously added.
- This sum is divided by *m*, where:

  *m* = 2 for binary items of 4-digit length or less

x *m* = 4 for binary items of 5-digit length or more: USAGE IS INDEX, USAGE IS
x POINTER, USAGE IS PROCEDURE-POINTER, USAGE IS OBJECT REFER-
x ENCE and COMPUTATIONAL-1 data items

x *m* = 8 for COMPUTATIONAL-2 data items.

- If the remainder (*r*) of this division is equal to zero, no slack bytes are required. If the remainder is not equal to zero, the number of slack bytes that must be added is equal to *m* - *r*.

These slack bytes are added to each record immediately following the elementary data item preceding the binary item. They are defined as if they constituted an item with a level number equal to that of the elementary item that immediately precedes the SYNCHRONIZED binary item, and are included in the size of the group that contains them.

For example:

```
01  FIELD-A.
  05  FIELD-B                   PICTURE X(5).
  05  FIELD-C.
    10  FIELD-D                 PICTURE XX.
   [10  SLACK-BYTES              PICTURE X.  INSERTED BY COMPILER]
    10  FIELD-E  COMPUTATIONAL  PICTURE S9(6) SYNC.

01  FIELD-L.
  05  FIELD-M                   PICTURE X(5).
  05  FIELD-N                   PICTURE XX.
 [05  SLACK-BYTES                PICTURE X.  INSERTED BY COMPILER]
  05  FIELD-O.
    10  FIELD-P COMPUTATIONAL   PICTURE S9(6) SYNC.
```

Slack bytes can also be added by the compiler when a group item is defined with an OCCURS clause and contains within it a SYNCHRONIZED binary data item. To determine whether slack bytes are to be added, the following action is taken:

- The compiler calculates the size of the group, including all the necessary slack bytes within a record.
- This sum is divided by the largest *m* required by any elementary item within the group.
- If *r* is equal to zero, no slack bytes are required. If *r* is not equal to zero, *m* - *r* slack bytes must be added.

The slack bytes are inserted at the end of each occurrence of the group item containing the OCCURS clause. For example, a record defined as follows will appear in storage, as shown, in Figure 8:

```
01 WORK-RECORD.
  05  WORK-CODE               PICTURE X.
  05  COMP-TABLE OCCURS 10 TIMES.
    10  COMP-TYPE             PICTURE X.
   [10  SLACK-BYTES           PIC XX.  INSERTED BY COMPILER]
    10  COMP-PAY              PICTURE S9(4)V99 COMP SYNC.
    10  COMP-HRS              PICTURE S9(3) COMP SYNC.
    10  COMP-NAME             PICTURE X(5).
```

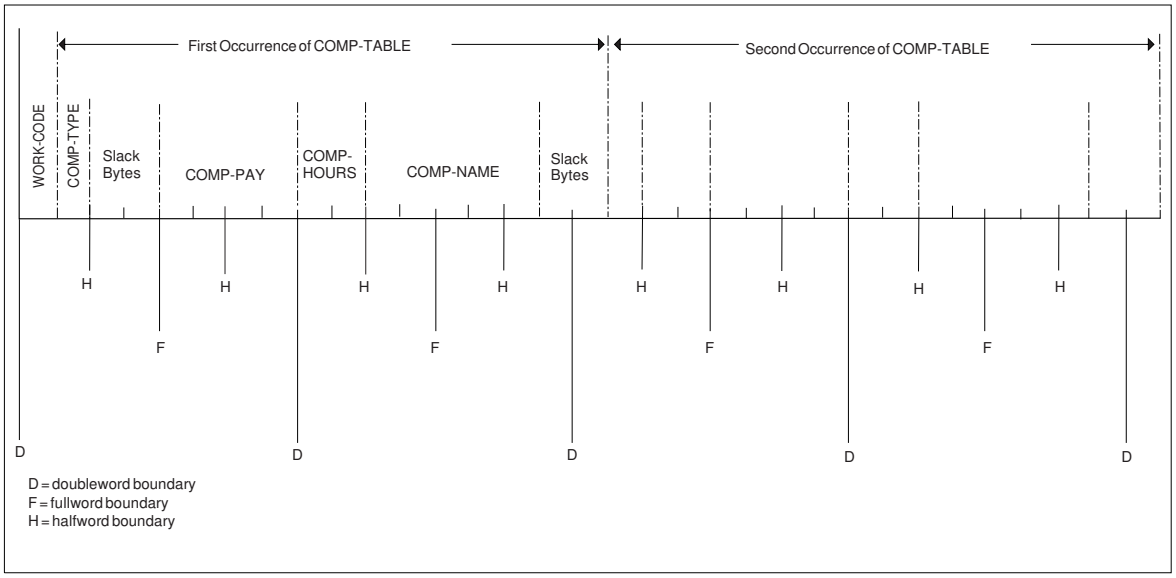



*Figure 8. Insertion of Slack Bytes within a Record*

In order to align COMP-PAY and COMP-HRS upon their proper boundaries, the compiler has added two slack bytes within the record.

In the example previous, without further adjustment, the second occurrence of COMP-TABLE would begin one byte before a doubleword boundary, and the alignment of COMP-PAY and COMP-HRS would not be valid for any occurrence of the table after the first. Therefore, the compiler must add slack bytes at the end of the group, as though the record had been written as follows:

```
01  WORK-RECORD.
  05  WORK-CODE               PICTURE X.
  05  COMP-TABLE OCCURS 10 TIMES.
    10  COMP-TYPE             PICTURE X.
   [10  SLACK-BYTES            PIC XX.  INSERTED BY COMPILER ]
    10  COMP-PAY              PICTURE S9(4)V99 COMP SYNC.
    10  COMP-HRS              PICTURE S9(3)  COMP SYNC.
    10  COMP-NAME             PICTURE X(5).
   [10  SLACK-BYTES            PIC XX.  INSERTED BY COMPILER]
```

In this example, the second (and each succeeding) occurrence of COMP-TABLE begins one byte beyond a doubleword boundary. The storage layout for the first occurrence of COMP-TABLE will now appear as shown in Figure 9.

## SYNCHRONIZED Clause

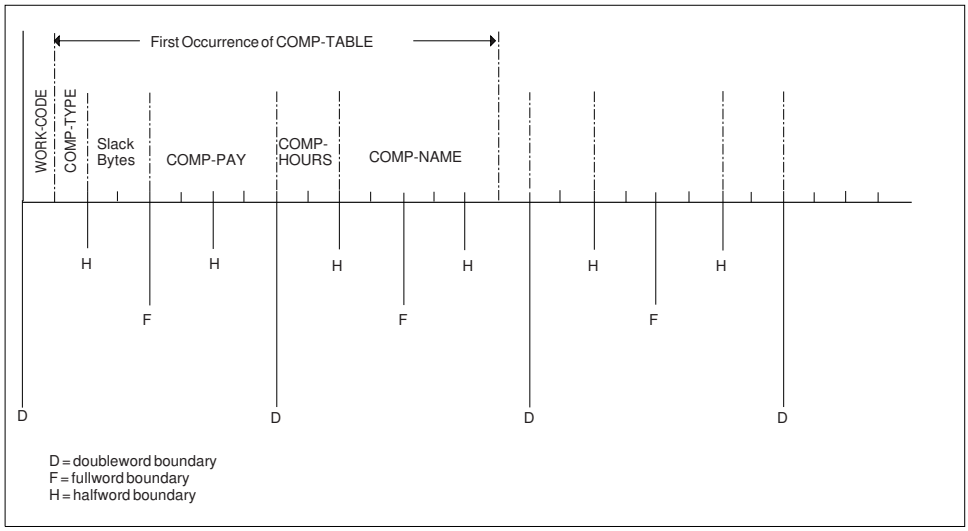



*Figure 9. Insertion of Slack Bytes between Records*

Each succeeding occurrence within the table will now begin at the same relative position as the first.

## Slack Bytes between Records

MVS/VM Under MVS and VM, if the file contains blocked logical records that are to be processed in a buffer, and any of the records contain binary entries for which the SYNCHRONIZED clause is specified, you can improve performance by adding any needed slack bytes between records for proper alignment.

The lengths of all the elementary data items in the record, including all slack bytes, are added. (For variable-length records under MVS and VM, it is necessary to add an additional 4 bytes for the count field.) The total is then divided by the highest value of *m* for any one of the elementary items in the record.

If *r* (the remainder) is equal to zero, no slack bytes are required. If *r* is not equal to zero, *m* - *r* slack bytes are required. These slack bytes can be specified by writing a level-02 FILLER at the end of the record.

To show the method of calculating slack bytes both within and between records, consider the following record description:

```
01  COMP-RECORD.
  05  A-1           PICTURE X(5).
  05  A-2           PICTURE X(3).
  05  A-3           PICTURE X(3).
  05  B-1           PICTURE S9999  USAGE COMP SYNCHRONIZED.
  05  B-2           PICTURE S99999 USAGE COMP SYNCHRONIZED.
  05  B-3           PICTURE S9999  USAGE COMP SYNCHRONIZED.
```

The number of bytes in A-1, A-2, and A-3 totals 11. B-1 is a 4-digit COMPUTATIONAL item and 1 slack byte must therefore be added before B-1. With this byte added, the number of bytes preceding B-2 totals 14. Because B-2 is a COMPUTATIONAL item of 5 digits in length, two slack bytes must be added before it. No slack bytes are needed before B-3.

The revised record description entry now appears as:

```
01  COMP-RECORD.
  05  A-1             PICTURE X(5).
  05  A-2             PICTURE X(3).
  05  A-3             PICTURE X(3).
 [05  SLACK-BYTE-1    PICTURE X.   INSERTED BY COMPILER]
  05  B-1             PICTURE S9999 USAGE COMP SYNCHRONIZED.
 [05  SLACK-BYTE-2    PICTURE XX.  INSERTED BY COMPILER]
  05  B-2             PICTURE S99999 USAGE COMP SYNCHRONIZED.
  05  B-3             PICTURE S9999  USAGE COMP SYNCHRONIZED.
```

There is a total of 22 bytes in COMP-RECORD, but, from the rules given in the preceding discussion, it appears that *m* = 4 and *r* = 2. Therefore, to attain proper alignment for blocked records, you must add 2 slack bytes at the end of the record.

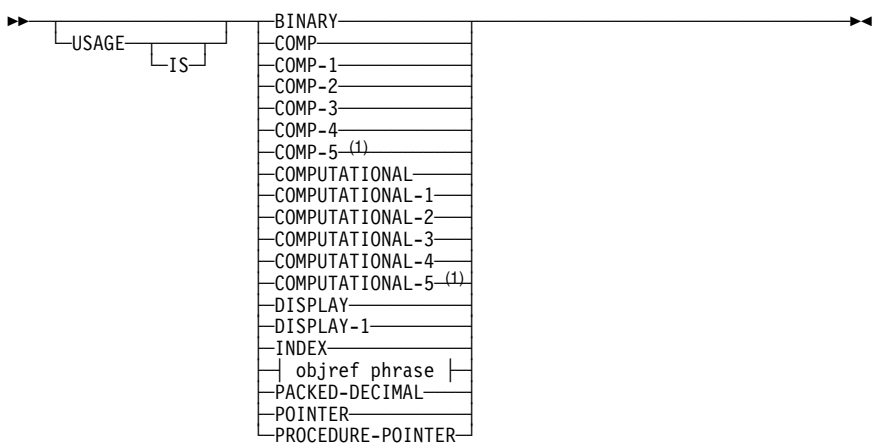
The final record description entry appears as:

```
01  COMP-RECORD.
  05  A-1              PICTURE X(5).
  05  A-2              PICTURE X(3).
  05  A-3              PICTURE X(3).
 [05  SLACK-BYTE-1     PICTURE X.   INSERTED BY COMPILER]
  05  B-1              PICTURE S9999 USAGE COMP SYNCHRONIZED.
 [05  SLACK-BYTE-2     PICTURE XX.  INSERTED BY COMPILER]
  05  B-2              PICTURE S99999 USAGE COMP SYNCHRONIZED.
  05  B-3              PICTURE S9999  USAGE COMP SYNCHRONIZED.
  05  FILLER           PICTURE XX.  [SLACK BYTES YOU ADD]
```

## USAGE Clause

The USAGE clause specifies the format of a data item in computer storage.

**Format 1**




**Note:**
[1] COMP-5 and COMPUTATIONAL-5 are only supported on AIX, OS/2, and Windows.

The USAGE clause can be specified for a data description entry with a level-number other than 66 or 88. However, if it is specified at the group level, it applies to each elementary item in the group. The usage of an elementary item must not contradict the usage of a group to which the elementary item belongs.

The USAGE clause specifies the format in which data is represented in storage. The format can be restricted if certain Procedure Division statements are used.

When the USAGE clause is not specified at either the group or elementary level, it is assumed that the usage is DISPLAY.

### Computational Items

A computational item is a value used in arithmetic operations. It must be numeric. If the USAGE of a group item is described with any of these items, the elementary items within the group have this usage.

The maximum length of a computational item is 18 decimal digits.

The PICTURE of a computational item can contain only:

9 One or more numeric character positions
S One operational sign
V One implied decimal point
P One or more decimal scaling positions

x COMPUTATIONAL-1 and COMPUTATIONAL-2 items (internal floating-point) cannot
x have PICTURE strings.

**BINARY**

Specified for binary data items. Such items have a decimal equivalent consisting of the decimal digits 0 through 9, plus a sign. Negative numbers are represented as the two's complement of the positive number with the same absolute value.

The amount of storage occupied by a binary item depends on the number of decimal digits defined in its PICTURE clause:

| Digits in PICTURE Clause | Storage Occupied |
|---|---|
| 1 through 4 | 2 bytes (halfword) |
| 5 through 9 | 4 bytes (fullword) |
| 10 through 18 | 8 bytes (doubleword) |

The operational sign for “big-endian” binary data (such as MVS and VM) is contained in the left most bit of the binary data. The operational sign for “little-endian” binary data is contained in the left most bit of the right most byte of the binary data.

x **Note:** BINARY, COMPUTATIONAL, and COMPUTATIONAL-4 data items can be
x affected by the BINARY and TRUNC compiler option specifications. For informa-
x tion on the effect of these compiler options, see the *IBM COBOL Programming*
x *Guide* for your platform. (BINARY is only applicable to AIX, OS/2, and Windows
x programs.)

**PACKED-DECIMAL**

Specified for internal decimal items. Such an item appears in storage in packed decimal format. There are 2 digits for each character position, except for the trailing character position, which is occupied by the low-order digit and the sign. Such an item can contain any of the digits 0 through 9, plus a sign, representing a value not exceeding 18 decimal digits.

The sign representation uses the same bit configuration as the 4-bit sign representation in zoned decimal fields. For details, see the *IBM COBOL Programming Guide* for your platform.

**COMPUTATIONAL or COMP (Binary)**

This is the equivalent of BINARY. The COMPUTATIONAL phrase is synonymous with BINARY.

x **COMPUTATIONAL-1 or COMP-1 (Floating-Point)**

x Specified for internal floating-point items (single precision). COMP-1 items are 4
x bytes long.

## USAGE Clause

x **COMPUTATIONAL-2 or COMP-2 (Long Floating-Point)**
x Specified for internal floating-point items (double precision). COMP-2 items are 8
x bytes long.

x **COMPUTATIONAL-3 or COMP-3 (Internal Decimal)**
x This is the equivalent of PACKED-DECIMAL.

x **COMPUTATIONAL-4 or COMP-4 (Binary)**
x This is the equivalent of BINARY.

x **COMPUTATIONAL-5 or COMP-5 (Native Binary)**
x WRKSTN Under AIX, OS/2, and Windows, this represents native binary data. In a
x COMP-5 binary data item, the total precision of the literal (integer and decimal
x positions) can be up to the maximum value that can be accommodated in the
x number of bytes allocated for the binary field (for example, 2, 4, or 8 bytes) rather
x than conforming to the maximum integer and decimal positions specified in the
x PICTURE clause. This is always the native binary data, independent of the
x BINARY compiler option.

## DISPLAY Phrase

The data item is stored in character form, 1 character for each 8-bit byte. This corresponds to the format used for printed output. DISPLAY can be explicit or implicit.

USAGE IS DISPLAY is valid for the following types of items:

- Alphabetic
- Alphanumeric
- Alphanumeric-edited
- Numeric-edited
x • External floating-point
- External decimal (numeric).

**Alphabetic**, **alphanumeric**, **alphanumeric-edited**, and **numeric-edited** items are discussed in “Data Categories and PICTURE Rules” on page 176.

**External Decimal Items** are sometimes referred to as **zoned decimal** items. Each digit of a number is represented by a single byte. The 4 high-order bits of each byte are zone bits; the 4 high-order bits of the low-order byte represent the sign of the item. The 4 low-order bits of each byte contain the value of the digit.

The maximum length of an external decimal item is 18 digits.

The PICTURE character-string of an external decimal item can contain only 9s; the operational-sign, S; the assumed decimal point, V; and one or more Ps.

## DISPLAY-1 Phrase

x The DISPLAY-1 phrase defines an item as DBCS.

## INDEX Phrase

A data item defined with the INDEX phrase is an **index data item**.

An **index data item** is a 4-byte elementary item (not necessarily connected with any table) that can be used to save index-name values for future reference. Through a SET statement, an index data item can be assigned an index-name value; such a value corresponds to the occurrence number in a table.

Direct references to an index data item can be made only in a SEARCH statement, a SET statement, a relation condition, the USING phrase of the Procedure Division header, or the USING phrase of the CALL statement.

x An index data item can be referred to directly in the USING phrase of an ENTRY state-
x ment.

An index data item can be part of a group item referred to in a MOVE statement or an input/output statement.

An index data item saves values that represent table occurrences, yet is not necessarily defined as part of any table. Thus, when it is referred to directly in a SEARCH or SET statement, or indirectly in a MOVE or input/output statement, there is no conversion of values when the statement is executed.

The USAGE IS INDEX clause can be written at any level. If a group item is described with the USAGE IS INDEX clause, the elementary items within the group are index data items; the group itself is not an index data item, and the group name cannot be used in SEARCH and SET statements or in relation conditions. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An index data item cannot be a conditional variable.

The JUSTIFIED, PICTURE, BLANK WHEN ZERO, SYNCHRONIZED, or VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

x SYNCHRONIZED can be used with USAGE IS INDEX to obtain efficient use of the
x index data item.

## x OBJECT REFERENCE Phrase

x A data item defined with the OBJECT REFERENCE phrase is an **object reference**.

x **class-name-1**

x An optional class name.

x You must declare class-name-1 in the REPOSITORY paragraph in the Configura-
x tion Section of the containing class or outermost program. If specified,
x class-name-1 indicates that data-name always refers to an object-instance of class
x class-name-1 or a class derived from class-name-1.

If class-name-1 is not specified, data-name can refer to an object of any class. In this case, data-name-1 is a "universal" object reference.

You can specify data-name-1 within a group item without affecting the semantics of the group item. There is no conversion of values or other special handling of the object references when statements are executed that operate on the group. The group continues to behave as an alphanumeric data item.

**METACLASS**
Indicates that the data-name always refers to a class object reference that is an instance of the metaclass of class-name-1 or of a metaclass derived from the metaclass of class-name-1.

You can use these object references to INVOKE methods that are defined in the metaclass.

The USAGE IS OBJECT REFERENCE clause can be used at any level except level 66 or 88. If a group item is described with the USAGE IS OBJECT REFERENCE clause, the elementary items within the group are object-reference data items. The group itself is not an object reference. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group that contains the item.

An object reference can be defined in any section of the data division of a class, method, or program, although it does not belong to any class or category. An object-reference data item can be used in only:

- A SET statement (Format 7 only)
- A relation condition
- An INVOKE statement
- The USING or RETURNING phrase of an INVOKE statement
- The USING or RETURNING phrase of a CALL statement
- A program Procedure Division or ENTRY statement USING or RETURNING phrase
- A method Procedure Division USING or RETURNING phrase

Object reference data items:

- Are ignored in CORRESPONDING operations
- Are unaffected by INITIALIZE statements
- Can be the subject or object of a REDEFINES clause
- Cannot be a conditional variable
- Can be written to a file (but upon subsequent reading of the record the content of the object reference is undefined)

A VALUE clause for an object-reference data item can contain only NULL or NULLS.

You can use the SYNCHRONIZED clause with USAGE IS OBJECT REFERENCE to obtain efficient alignment of the object-reference data item.

The JUSTIFIED, PICTURE, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items defined with the USAGE IS OBJECT REFERENCE clause.

## POINTER Phrase

A data item defined with USAGE IS POINTER is a **pointer data item**. A pointer data item is a 4-byte elementary item,

You can use pointer data items to accomplish limited base addressing. Pointer data items can be compared for equality or moved to other pointer items.

A pointer data item can only be used:

- In a SET statement (Format 5 only)
- In a relation condition
- In the USING phrase of a CALL statement, an ENTRY statement, or the Procedure Division header.

The USAGE IS POINTER clause can be written at any level except level 88. If a group item is described with the USAGE IS POINTER clause, the elementary items within the group are pointer data items; the group itself is not a pointer data item and cannot be used in the syntax where a pointer data item is allowed. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

Pointer data items can be part of a group that is referred to in a MOVE statement or an input/output statement. However, if a pointer data item is part of a group, there is no conversion of values when the statement is executed.

A pointer data item can be the subject or object of a REDEFINES clause.

SYNCHRONIZED can be used with USAGE IS POINTER to obtain efficient use of the pointer data item.

A VALUE clause for a pointer data item can contain only NULL or NULLS.

A pointer data item cannot be a conditional variable.

A pointer data item does not belong to any class or category.

The JUSTIFIED, PICTURE, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items defined with the USAGE IS POINTER clause.

Pointer data items are ignored in CORRESPONDING operations.

A pointer data item can be written to a data set, but, upon subsequent reading of the record containing the pointer, the address contained can no longer represent a valid pointer.

**Note:** USAGE IS POINTER is implicitly specified for the ADDRESS OF special register. For more information see the *IBM COBOL Programming Guide* for your platform.

## PROCEDURE-POINTER Phrase

A procedure-pointer data item can contain the address of a procedure entry point. Procedure-pointer data items can be compared for equality or moved to other procedure-pointer data items.

MVS/VM Under MVS and VM, a procedure-pointer data item is an 8-byte elementary item.

WRKSTN Under AIX, OS/2, and Windows, a procedure-pointer data item is a 4-byte elementary item.

The entry point for a procedure-pointer data item can be:

- The primary entry point of a COBOL program as defined by the PROGRAM-ID statement of the outermost program of a compilation unit; it must not be the PROGRAM-ID of a nested program.
- An alternate entry point of a COBOL program as defined by a COBOL ENTRY statement
- An entry point in a non-COBOL program.

The entry point address and code address are contained in the first word. The second word is binary zero.

A procedure-pointer data item can only be used:

- In a SET statement (Format 6 only)
- In a CALL statement
- In a relation condition
- In the USING phrase of an ENTRY statement or the Procedure Division header

The USAGE IS PROCEDURE-POINTER clause can be written at any level except level 88. If a group item is described with the USAGE IS PROCEDURE-POINTER clause, the elementary items within the group are procedure-pointer data items; the group itself is not a procedure-pointer and cannot be used in the syntax where a procedure-pointer data item is allowed. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

Procedure-pointer data items can be part of a group that is referred to in a MOVE statement or an input/output statement. However, there is no conversion of values when the statement is executed. If a procedure-pointer data item is written to a data set, subsequent reading of the record containing the procedure-pointer can result in an invalid value in the procedure-pointer.

A procedure-pointer data item can be the subject or object of a REDEFINES clause.

SYNCHRONIZED can be used with USAGE IS PROCEDURE-POINTER to obtain efficient alignment of the procedure-pointer data item.

x The GLOBAL, EXTERNAL, and OCCURS clause can be used with USAGE IS
x PROCEDURE-POINTER.

x A VALUE clause for a procedure-pointer data item can contain only NULL or NULLS.

x The JUSTIFIED, PICTURE, and BLANK WHEN ZERO clauses cannot be used to
x describe group or elementary items defined with the USAGE IS
x PROCEDURE-POINTER clause.

x A procedure-pointer data item cannot be a conditional variable.

x A procedure-pointer data item does not belong to any class or category.

x Procedure-pointer data items are ignored in CORRESPONDING operations.

## VALUE Clause

The VALUE clause specifies the initial contents of a data item or the value(s) associated with a condition name. The use of the VALUE clause differs depending on the Data Division section in which it is specified.

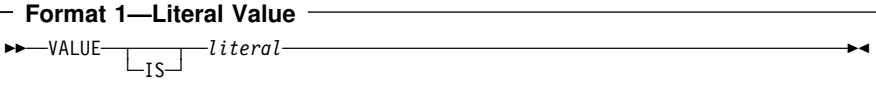
x In the class Working-Storage Section, the VALUE clause can only be used in condition-
x name entries.

x As an IBM extension, in the File and Linkage Sections, if the VALUE clause is used in
x entries other than condition-name entries, the VALUE clause is treated as a comment.

In the Working-Storage Section, the VALUE clause can be used in condition-name entries, or in specifying the initial value of any data item. The data item assumes the specified value at the beginning of program execution. If the initial value is not explicitly specified, it is unpredictable.

### Format 1

**Format 1—Literal Value**




Format 1 specifies the initial value of a data item. Initialization is independent of any BLANK WHEN ZERO or JUSTIFIED clause specified.

A format 1 VALUE clause specified in a data description entry that contains or is subordinate to an OCCURS clause causes every occurrence of the associated data item to be assigned the specified value. Each structure that contains the DEPENDING ON phrase of the OCCURS clause is assumed to contain the maximum number of occurrences for the purposes of VALUE initialization.

## VALUE Clause

The VALUE clause must not be specified for a data description entry that contains, or is subordinate to, an entry containing either an EXTERNAL or a REDEFINES clause. This rule does not apply to condition-name entries.

If the VALUE clause is specified at the group level, the literal must be a nonnumeric literal or a figurative constant. The group area is initialized without consideration for the subordinate entries within this group. In addition, the VALUE clause must not be specified for subordinate entries within this group.

For group entries, the VALUE clause must not be specified if the entry also contains any of the following clauses: JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE DISPLAY).

The VALUE clause must not conflict with other clauses in the data description entry, or in the data description of this entry's hierarchy.

x Any VALUE clause associated with COMPUTATIONAL-1 or COMPUTATIONAL-2
x (internal floating-point) items must specify a floating-point literal. The condition-name
x VALUE phrase must also specify a floating-point literal. In addition, the figurative con-
x stant ZERO and both integer and decimal forms of the zero literal can be specified in a
x floating-point VALUE clause or condition-name VALUE phrase.

x For information on floating-point literal values, see “Rules for Floating-point Literal
x Values:” on page 27.

x A VALUE clause cannot be specified for external floating-point items.

x A VALUE clause associated with a DBCS item must contain a DBCS literal or the figu-
x rative constant SPACE.

x A data item cannot contain a VALUE clause if the prior data item contains a OCCURS
x clause with the DEPENDING ON phrase.

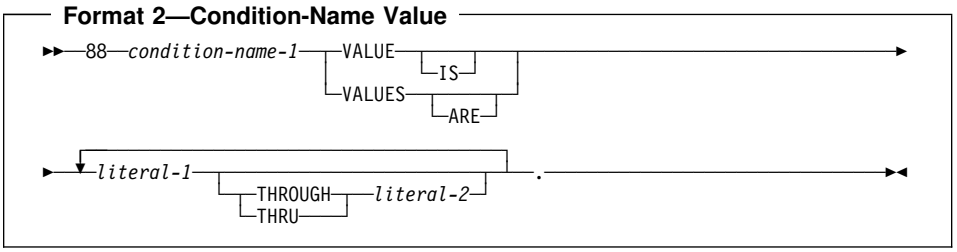
### Rules for Literal Values:

- Wherever a literal is specified, a figurative constant can be substituted.
- If the item is numeric, all VALUE clause literals must be numeric. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves, with one additional restriction: The literal must not have a value that requires truncation of nonzero digits. If the literal is signed, the associated PICTURE character-string must contain a sign symbol (S).
- All numeric literals in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause for that item. For example, for PICTURE 99PPP, the literal must be within the range 1000 through 99000, or zero. For PICTURE PPP99, the literal must be within the range 0.00000 through 0.00099.
- If the item is an elementary or group alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited item, all VALUE clause literals must be nonnumeric literals. The literal is aligned according to the alphanumeric alignment rules, with

one additional restriction: The number of characters in the literal must not exceed the size of the item.

- The functions of the editing characters in a PICTURE clause are ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item. Therefore, any editing characters must be included in the literal. For example, if the item is defined as PICTURE +999.99 and the value is to be +12.34, then the VALUE clause should be specified as VALUE "+012.34".

## Format 2




This format associates a value, values, and/or range(s) of values with a condition-name. Each such condition-name requires a separate level-88 entry. Level-number 88 and condition-name are not part of the Format 2 VALUE clause itself. They are included in the format only for clarity.

**condition-name-1**
A user-specified name that associates a value with a conditional variable. If the associated conditional variable requires subscripts or indexes, each procedural reference to the condition-name must be subscripted or indexed as required for the conditional variable.

Condition-names are tested procedurally in condition-name conditions (see "Conditional Expressions" on page 225).

**literal-1**
When literal-1 is specified alone, the condition-name is associated with a single value.

**literal-1 THROUGH literal-2**
The condition-name is associated with at least one range of values. Whenever the THROUGH phrase is used, literal-1 must be less than literal-2.

x In the VALUE clause of a data description entry (Format 2), all the literals specified for
x the THROUGH phrase must be DBCS literals if the associated conditional variable is a
x DBCS data item. The figurative constants SPACE and SPACES can be used as DBCS
x literals.

x MVS/VM Under MVS and VM, the range of DBCS literals specified for the THROUGH
x phrase is based on the binary collating sequence of the hexadecimal values of the
x DBCS characters.

x WRKSTN Under AIX, OS/2, and Windows, the range of nonnumeric literals or DBCS
x literals specified for the THROUGH phrase is based on the collating sequence indicated
x by the locale (except for single-byte character comparisons when a non-NATIVE col-
x lating sequence is in effect). For more information on locale, see Appendix F, "Locale
x Considerations (Workstation Only)" on page 543.

### Rules for Condition-Name Values:

- The VALUE clause is required in a condition-name entry, and must be the only clause in the entry. Each condition-name entry is associated with a preceding conditional variable. Thus, every level-88 entry must always be preceded either by the entry for the conditional variable, or by another level-88 entry when several condition-names apply to one conditional variable. Each such level-88 entry implicitly has the PICTURE characteristics of the conditional variable.
- The key words THROUGH and THRU are equivalent.

  The condition-name entries associated with a particular conditional variable must immediately follow the conditional variable entry. The conditional variable can be any elementary data description entry except another condition-name, a RENAMES clause (level-66 item), or an item with USAGE IS INDEX.

x The conditional variable cannot be an item with USAGE IS POINTER, USAGE IS
x PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE.

  A condition-name can be associated with a group item data description entry. In this case:

  - The condition-name value must be specified as a nonnumeric literal or figurative constant.
  - The size of the condition-name value must not exceed the sum of the sizes of all the elementary items within the group.
  - No element within the group can contain a JUSTIFIED or SYNCHRONIZED clause.
  - No USAGE other than DISPLAY can be specified within the group.

x USAGE other than USAGE IS DISPLAY can be specified within the group.

  Condition-names can be specified both at the group level and at subordinate levels within the group.

  The relation test implied by the definition of a condition-name at the group level is performed in accordance with the rules for comparison of nonnumeric operands, regardless of the nature of elementary items within the group.

x The VALUE clause is allowed for internal floating-point data items.

x The VALUE clause is allowed for DBCS data items. Relation tests for DBCS data
x items are performed according to the rules for comparison of DBCS items. These
x rules can be found in "Comparison of DBCS Operands" on page 237.

  A space, a separator comma, or a separator semicolon, must separate successive operands.

Each entry must end with a separator period.

- The type of literal in a condition-name entry must be consistent with the data type of its conditional variable. In the following example:
  - CITY-COUNTY-INFO, COUNTY-NO, and CITY are conditional variables.

    The PICTURE associated with COUNTY-NO limits the condition-name value to a 2-digit numeric literal.

    The PICTURE associated with CITY limits the condition-name value to a 3-character nonnumeric literal.
  - The associated condition-names are level-88 entries.

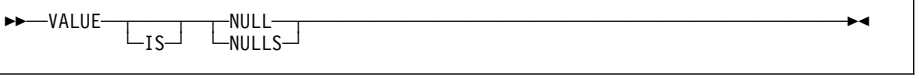
    Any values for the condition-names associated with CITY-COUNTY-INFO cannot exceed 5 characters.

    Because this is a group item, the literal must be nonnumeric.

```
05  CITY-COUNTY-INFO.
    88  BRONX                  VALUE "03NYC".
    88  BROOKLYN               VALUE "24NYC".
    88  MANHATTAN              VALUE "31NYC".
    88  QUEENS                 VALUE "41NYC".
    88  STATEN-ISLAND          VALUE "43NYC".
  10  COUNTY-NO                PICTURE 99.
      88  DUTCHESS             VALUE 14.
      88  KINGS                VALUE 24.
      88  NEW-YORK             VALUE 31.
      88  RICHMOND             VALUE 43.
  10  CITY                     PICTURE X(3).
      88  BUFFALO              VALUE "BUF".
      88  NEW-YORK-CITY        VALUE "NYC".
      88  POUGHKEEPSIE         VALUE "POK".
05  POPULATION...
```

x ## Format 3

x **Format 3—NULL Value**

x 
x 

x This format assigns an invalid address as the initial value of an item defined as USAGE
x IS POINTER or USAGE IS PROCEDURE-POINTER. It also assigns an invalid object
x reference as the initial value of an item defined as USAGE IS OBJECT REFERENCE.

x VALUE IS NULL can only be specified for elementary items described implicitly or
x explicitly as USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS
x OBJECT REFERENCE.

## VALUE Clause

# Part 6. Procedure Division


**Procedure Division Structure** . . . . . 215
x Requirements for a Method Procedure Division . . . . . 216
The Procedure Division Header . . . . . 217
Declaratives . . . . . 220
Procedures . . . . . 221
Arithmetic Expressions . . . . . 223
Conditional Expressions . . . . . 225
Statement Categories . . . . . 245
Statement Operations . . . . . 249

**Procedure Division Statements** . . . . . 261
ACCEPT Statement . . . . . 261
ADD Statement . . . . . 266
ALTER Statement . . . . . 268
CALL Statement . . . . . 270
CANCEL Statement . . . . . 277
CLOSE Statement . . . . . 279
COMPUTE Statement . . . . . 283
CONTINUE Statement . . . . . 285
DELETE Statement . . . . . 286
DISPLAY Statement . . . . . 288
DIVIDE Statement . . . . . 291
ENTRY Statement . . . . . 294
EVALUATE Statement . . . . . 295
EXIT Statement . . . . . 299
x EXIT METHOD Statement . . . . . 300
EXIT PROGRAM Statement . . . . . 301
GOBACK Statement . . . . . 302
GO TO Statement . . . . . 303
IF Statement . . . . . 305
INITIALIZE Statement . . . . . 307
INSPECT Statement . . . . . 310
x INVOKE Statement . . . . . 319
MERGE Statement . . . . . 327
MOVE Statement . . . . . 333
MULTIPLY Statement . . . . . 338
OPEN Statement . . . . . 340
PERFORM Statement . . . . . 346
READ Statement . . . . . 358
RELEASE Statement . . . . . 366
RETURN Statement . . . . . 368
REWRITE Statement . . . . . 370
SEARCH Statement . . . . . 374
SET Statement . . . . . 381
SORT Statement . . . . . 388


© Copyright IBM Corp. 1991, 1996

START Statement . . . . . 395
STOP Statement . . . . . 398
STRING Statement . . . . . 400
SUBTRACT Statement . . . . . 405
UNSTRING Statement . . . . . 407
WRITE Statement . . . . . 415

# Procedure Division Structure

The Procedure Division is optional in a COBOL source program, class definition, and method definition.

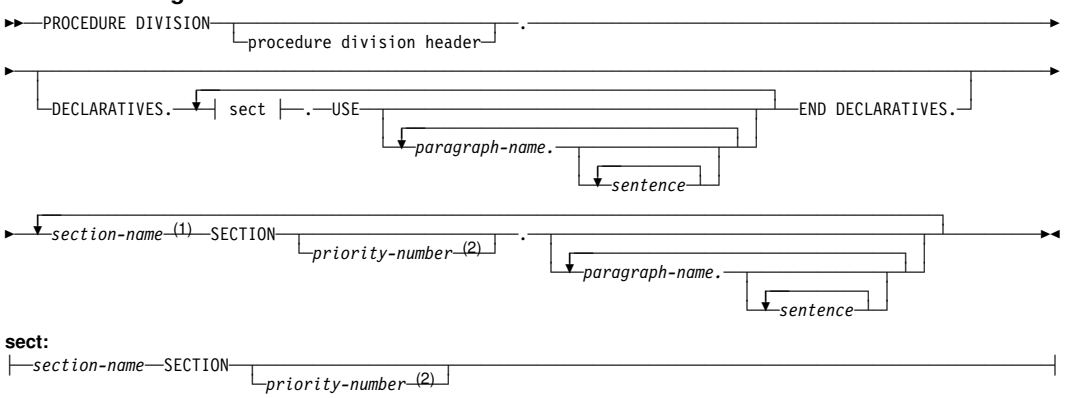
**Program Procedure Division**
A program Procedure Division consists of optional declaratives, and procedures that contain sections and/or paragraphs, sentences, and statements.

**Class Procedure Division**
The class Procedure Division contains only method definitions. All methods introduced in a COBOL class compilation unit must be defined in that compilation unit's Procedure Division.

**Method Procedure Division**
A method Procedure Division consists of optional declaratives, and procedures that contain sections and/or paragraphs, sentences, and statements. A method can INVOKE other methods, be recursively INVOKEd, and issue a CALL to a program. A method Procedure Division cannot contain nested programs or methods.

For additional details on a method Procedure Division, see “Requirements for a Method Procedure Division” on page 216.

**Format—Program and Method Procedure Division**

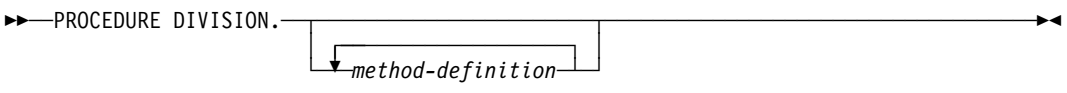



**Notes:**

1 As an IBM extension, section-name can be omitted. If you omit section-name, paragraph-name can be omitted.

2 Priority-numbers are not valid for methods, recursive programs, or (on AIX, OS/2, and Windows) programs compiled with the THREAD option.

**Format—Class Procedure Division**




© Copyright IBM Corp. 1991, 1996

## Requirements for a Method Procedure Division

When using a Method Procedure Division, you need to know that:

- You can use the EXIT METHOD statement or the GOBACK statement to return control to the invoking method or program. An implicit EXIT METHOD statement is generated as the last statement of every method procedure division.

  For details on the EXIT METHOD statement, see "EXIT METHOD Statement" on page 300.
- You can use the STOP RUN statement (which terminates the run unit) in a method.
- You can use the RETURN-CODE special register within a method Procedure Division to access return codes from CALLed subprograms, but the RETURN-CODE value is not returned to the invoker of the current method. Use the Procedure Division RETURNING data name to return a value to the invoker of the current method. For details, see the discussion of RETURNING data-name-2 under "The Procedure Division Header" on page 217.

You cannot specify the following statements in a method PROCEDURE DIVISION:

- ALTER
- ENTRY
- EXIT PROGRAM
- GO TO without a specified procedure name
- SEGMENTATION
- USE FOR DEBUGGING

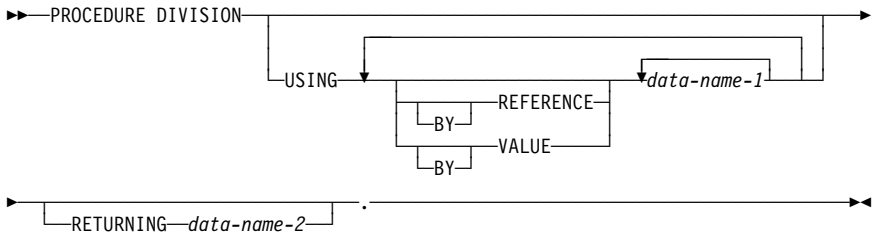
The following special registers are allocated on a per-invocation basis for methods; thus, they are in initial state on each method entry.

- ADDRESS OF (for each record in the Linkage Section)
- RETURN-CODE
- SORT-CONTROL
- SORT-CORE-SIZE
- SORT-FILE-SIZE
- SORT-MESSAGE
- SORT-MODE-SIZE
- SORT-RETURN
- TALLY

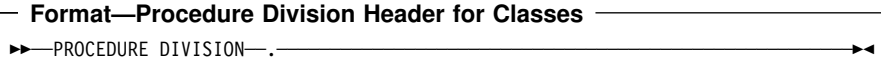
## The Procedure Division Header

The Procedure Division, if specified, is identified by one of the following headers, depending on whether you are defining a program, method, or class.

**Format—Procedure Division Header for Programs and Methods**




**Format—Procedure Division Header for Classes**




**USING**

The USING phrase makes data items defined in a calling program available to a called subprogram or an invoked method.

Only specify the USING phrase if the program is invoked by a CALL statement or a method is invoked by the INVOKE statement and the CALL or INVOKE statement includes a USING phrase.

The USING phrase is valid in the Procedure Division header of a called subprogram entered at the beginning of the nondeclaratives portion; each USING identifier must be defined as a level-01 or level-77 item in the Linkage Section of the called subprogram or invoked method; it must not contain a REDEFINES clause.

A data item in the USING phrase of the Procedure Division header can have a REDEFINES clause in its data description entry.

In a called subprogram entered at the first executable statement following an ENTRY statement, the USING option is valid in the ENTRY statement; each USING identifier must be defined as a level-01 or level-77 item in the Linkage Section of the called subprogram or invoked method. In a calling program, the USING phrase is valid for the CALL or INVOKE statement; each USING identifier must be defined as a level-01, level-77, or an elementary item in the Data Division.

Each USING identifier in a calling program can be a data item of any level in the Data Division.

It is possible to call from non-COBOL programs or pass user parameters from a system command to a COBOL main program.

The order of appearance of USING identifiers in both calling and called subprograms or invoking and invoked methods, determines the correspondence of single sets of data available to both programs. The correspondence is positional and not

by name. For calling and called subprograms, corresponding identifiers must contain the same number of characters, although their data descriptions need not be the same. For invoking and invoked methods, see "Conformance Requirements for USING Phrase" on page 323.

For index-names, no correspondence is established; index-names in calling and called programs or invoking and invoked methods always refer to separate indexes.

The identifiers specified in a CALL USING or INVOKE USING statement name data items available to the calling program or invoking method that can be referred to in the called program or invoked method; a given identifier can appear more than once. These items are defined in any Data Division section.

As an IBM extension, an identifier can appear more than once in a Procedure Division USING phrase. The last value passed to it by a CALL USING or INVOKE USING statement is used. The BY REFERENCE or BY VALUE phrase applies to all parameters that follow until overridden by another BY REFERENCE or BY VALUE phrase.

**BY REFERENCE**

When a CALL or INVOKE argument is passed BY CONTENT or BY REFERENCE, BY REFERENCE must be specified or implied for the corresponding formal parameter on the PROCEDURE/ENTRY USING phrase.

BY REFERENCE is the default if neither BY REFERENCE or BY VALUE is specified.

If the reference to the corresponding data item in the CALL or INVOKE statement declares the parameter to be passed BY REFERENCE (explicit or implicit), the object program executes as if each reference to a USING identifier in the called subprogram or invoked method Procedure Division is replaced by a reference to the corresponding USING identifier in the calling program or invoked method.

If the reference to the corresponding data item in the CALL or INVOKE statement declares the parameter to be passed BY CONTENT, the value of the item is moved when the CALL or INVOKE statement is executed and placed into a system-defined storage item possessing the attributes declared in the Linkage Section for data-name-1. The data description of each parameter in the BY CONTENT phrase of the CALL or INVOKE statement must be the same, meaning no conversion or extension or truncation, as the data description of the corresponding parameter in the USING phrase of the Procedure Division header.

**BY VALUE**

If the reference to the corresponding data item in the CALL or INVOKE statement declares the parameter to be passed BY VALUE, then the value of the argument is passed, not a reference to the sending data item. Since CALLed subprograms and INVOKEd methods have access only to a temporary copy of the sending data item, any modifications made to the formal parameters corresponding to the BY VALUE argument do not affect the argument.

Examples illustrating these concepts can be found in *IBM COBOL Programming Guide* for your platform.

**RETURNING data-name-2**

Is the RETURNING phrase identifier. It specifies a data item to be returned as a program or method result. You must define data-name-2 as a either level 01 or 77 entry in the Linkage Section.

Data-name-2 is an output-only parameter. When a program or method returns to its invoker, the value in data-name-2 is implicitly stored into the identifier specified in the CALL RETURNING phrase or the INVOKE RETURNING phrase, as described in "CALL Statement" on page 270 or "INVOKE Statement" on page 319. If, in the execution of a Procedure Division statement, data-name-2 is used as a sending item before being used as a receiving item, the results are undefined.

When you specify Procedure Division RETURNING data-name-2, the RETURN-CODE special register can be used within the PROCEDURE DIVISION only as a means of accessing return codes from CALLed subprograms. The RETURN-CODE value is not returned to the caller of the current program (the value in data-name-2 is).

If the CALLed subprogram is COBOL, the CALLed COBOL program must specify the RETURNING phrase on its Procedure Division header. In addition, data-name-2 and the corresponding RETURNING identifier in the target program must have the same PICTURE, USAGE, SIGN, SYNCHRONIZE, JUSTIFIED, and BLANK WHEN ZERO clauses (except that currency signs can differ and periods and commas can be interchanged due to the DECIMAL POINT IS COMMA clause).

Do not use the Procedure Division RETURNING phrase in:

- Programs that contain the ENTRY statement
- Nested programs
- Main programs— results of specifying Procedure Division RETURNING on a main program are undefined. You should only specify the Procedure Division RETURNING phrase on called subprograms. For main programs, use the RETURN-CODE special register to return a value to the operating environment.
- MVS/VM Under MVS and VM, on programs that use CEEPIPI—results of specifying Procedure Division RETURNING on programs that are called with the Language Environment preinitialization service (CEEPIPI) are undefined.

## Declaratives

x Data items defined in the Linkage Section of the called program or invoked method, can be referenced within the Procedure Division of that program if, and only if, they satisfy one of the following conditions:

- x They are operands of the USING phrase of the Procedure Division header or the x ENTRY statement
- x They are operands of SET ADDRESS OF, CALL...BY REFERENCE ADDRESS x OF, or INVOKE...BY REFERENCE ADDRESS OF
- They are defined with a REDEFINES or RENAMES clause, the object of which satisfies the above conditions
- They are items subordinate to any item that satisfies the condition in the rules above
- They are condition-names or index-names associated with data items that satisfy any of the above conditions.

---

## Declaratives

Declaratives provide one or more special-purpose sections that are executed when an exceptional condition occurs.

When Declarative Sections are specified, they must be grouped at the beginning of the Procedure Division, and the entire Procedure Division must be divided into sections.

Each Declarative Section starts with a USE statement that identifies the section's function; the series of procedures that follow specify what actions are to be taken when the exceptional condition occurs. Each Declarative Section ends with another section-name followed by a USE statement, or with the key words END DECLARATIVES. See "USE Statement" on page 508 for more information on the USE statement.

The entire group of Declarative Sections is preceded by the key word DECLARATIVES, written on the line after the Procedure Division header; the group is followed by the key words END DECLARATIVES. The key words DECLARATIVES and END DECLARATIVES must each begin in Area A and be followed by a separator period. No other text can appear on the same line.

In the declaratives part of the Procedure Division, each section header must be followed by a separator period, and must be followed by a USE statement, followed by a separator period. No other text can appear on the same line.

There are three formats of the USE statement:

1. EXCEPTION declarative (see "USE Statement" on page 508)
2. DEBUGGING declarative (see "USE Statement" on page 508)
3. x LABEL declarative (see "USE Statement" on page 508).

The USE statement itself is never executed; instead, the USE statement defines the conditions that execute the succeeding procedural paragraphs, which specify the

actions to be taken. After the procedure is executed, control is returned to the routine that activated it.

Within a declarative procedure, except for the USE statement itself, there must be no reference to any nondeclarative procedure.

x As IBM extensions, the following apply to declarative procedures:

x For AIX, OS/2, Windows, MVS, and VM:

x • A declarative procedure can be performed from a nondeclarative procedure.

x Additionally for MVS and VM:

x • A nondeclarative procedure can be performed from a declarative procedure.

x • A declarative procedure can be referenced in a GO TO statement in a declar-
x ative procedure.

x • A nondeclarative procedure can be referenced in a GO TO statement in a
x declarative procedure.

Within a declarative procedure, no statement should be included that would cause the execution of a USE procedure that had been previously invoked and had not yet returned control to the invoking routine.

x You can include a statement that executes a previously invoked USE procedure that is
x still in control. However, to avoid an infinite loop, you must be sure there is an eventual
x exit at the bottom.

The declarative procedure is exited when the last statement in the procedure is executed.

## Procedures

Within the Procedure Division, a **procedure** consists of:

- A **section** or a group of sections
- A **paragraph** or group of paragraphs

A **procedure-name** is a user-defined name that identifies a section or a paragraph.

**Section**
A **section-header** optionally followed by one or more paragraphs.

**Section-header**
A **section-name** followed by the key word SECTION, optionally followed, by a **priority-number**, followed by a separator period.

If there are declaratives (format-1), a section-header is not required in the Procedure Division.

**Section-name**
A user-defined word that identifies a section. A referenced section-name, because it cannot be qualified, must be unique within the program in which it is defined.

**Priority-number**
x An integer or a positive signed numeric literal ranging in value from 0 through 99.

Sections in the declaratives portion must contain priority numbers in the range of 0 through 49.

x You cannot specify priority-numbers:

x • In a method definition
x • In a program that is declared with the RECURSIVE attribute
x • In a program that specifies the THREAD compiler option (Workstation
x only)

A section ends immediately before the next section header, or at the end of the Procedure Division, or, in the declaratives portion, at the key words END DECLARATIVES.

**Paragraph**
A **paragraph-name** followed by a separator period, optionally followed by one or more sentences.

**Note:** Paragraphs must be preceded by a period because paragraphs always follow either the ID Division Header, a Section, or another paragraph, all of which must end with a period.

**Paragraph-name**
A user-defined word that identifies a paragraph. A paragraph-name, because it can be qualified, need not be unique.

x If there are no declaratives (format-2), a paragraph-name is not required in the
x Procedure Division.

A paragraph ends immediately before the next paragraph-name or section header, or at the end of the Procedure Division, or, in the declaratives portion, at the key words END DECLARATIVES.

x As an IBM extension, all paragraphs do not need to be contained within sections,
x even if one or more paragraphs are so contained.

**Sentence**
One or more **statements** terminated by a separator period.

**Statement**
A syntactically valid combination of **identifiers** and symbols (literals, relational-operators, and so forth) beginning with a COBOL verb.

**identifier**
The word or words necessary to make unique reference to a data item, optionally including qualification, subscripting, indexing, and

reference-modification. In any Procedure Division reference (except the class test), the contents of an identifier must be compatible with the class specified through its PICTURE clause, or results are unpredictable.

Execution begins with the first statement in the Procedure Division, excluding declaratives. Statements are executed in the order in which they are presented for compilation, unless the statement rules dictate some other order of execution.

The end of the Procedure Division is indicated by one of the following:

- An Identification Division header, which indicates the start of a nested source program
- The END PROGRAM header
- The physical end of the program; that is, the physical position in a source program after which no further source program lines occur

## Arithmetic Expressions

Arithmetic expressions are used as operands of certain conditional and arithmetic statements.

An arithmetic expression can consist of any of the following:

1. An identifier described as a numeric elementary item (including numeric functions)
2. A numeric literal
3. The figurative constant ZERO
4. Identifiers and literals, as defined in items 1, 2, and 3, separated by arithmetic operators
5. Two arithmetic expressions, as defined in items 1, 2, 3, and/or 4, separated by an arithmetic operator
6. An arithmetic expression, as defined in items 1, 2, 3, 4, and/or 5, enclosed in parentheses.

Any arithmetic expression can be preceded by a unary operator.

Identifiers and literals appearing in arithmetic expressions must represent either numeric elementary items or numeric literals on which arithmetic can be performed.

If an exponential expression is evaluated as both a positive and a negative number, the result will always be the positive number. The square root of 4, for example,

```
4 ** 0.5 (the square root of 4)
```

is evaluated as +2 and -2. IBM COBOL always returns +2.

If the value of an expression to be raised to a power is zero, the exponent must have a value greater than zero. Otherwise, the size error condition exists. In any case where no real number exists as the result of the evaluation, the size error condition exists.

## Arithmetic Expressions

## Arithmetic Operators

Five binary arithmetic operators and two unary arithmetic operators (Table 15) can be used in arithmetic expressions. They are represented by specific characters that must be preceded and followed by a space.

*Table 15. Binary and Unary Operators*

| Binary Operator | Meaning | Unary Operator | Meaning |
|---|---|---|---|
| + | Addition | + | Multiplication by +1 |
| – | Subtraction | – | Multiplication by –1 |
| * | Multiplication | | |
| / | Division | | |
| ** | Exponentiation | | |

**Note:** Exponents in fixed-point exponential expressions cannot contain more than 9 digits. The compiler will truncate any exponent with more than 9 digits. In this case, the compiler will issue a diagnostic message if the exponent is a literal or constant; if the exponent is a variable or data-name, a diagnostic is issued at run-time.

Parentheses can be used in arithmetic expressions to specify the order in which elements are to be evaluated.

Expressions within parentheses are evaluated first. When expressions are contained within a nest of parentheses, evaluation proceeds from the least inclusive to the most inclusive set.

When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchic order is implied:

1. Unary operator
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction.

Parentheses either eliminate ambiguities in logic where consecutive operations appear at the same hierarchic level or modify the normal hierarchic sequence of execution when this is necessary. When the order of consecutive operations at the same hierarchic level is not completely specified by parentheses, the order is from left to right.

An arithmetic expression can begin only with a left parenthesis, a unary operator, or an operand (that is, an identifier or a literal). It can end only with a right parenthesis or an operand. An arithmetic expression must contain at least one reference to an identifier or a literal.

There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression, with each left parenthesis placed to the left of its corresponding right parenthesis.

If the first operator in an arithmetic expression is a unary operator, it must be immediately preceded by a left parenthesis if that arithmetic expression immediately follows an identifier or another arithmetic expression.

Table 16 shows permissible arithmetic symbol pairs. An arithmetic symbol pair is the combination of two such symbols in sequence. In the figure:

Yes indicates a permissible pairing.
No indicates that the pairing is not permitted.

*Table 16. Valid Arithmetic Symbol Pairs*

| | Second Symbol | | | | |
|---|---|---|---|---|---|
| **First Symbol** | **Identifier or Literal** | *** / ** + –** | **Unary + or Unary –** | **(** | **)** |
| **Identifier or Literal** | No | Yes | No | No | Yes |
| *** / ** + –** | Yes | No | Yes | Yes | No |
| **Unary + or Unary –** | Yes | No | No | Yes | No |
| **(** | Yes | No | Yes | Yes | No |
| **)** | No | Yes | No | No | Yes |

## Conditional Expressions

A conditional expression causes the object program to select alternative paths of control, depending on the truth value of a test. Conditional expressions are specified in EVALUATE, IF, PERFORM, and SEARCH statements.

A conditional expression can be specified in either simple conditions or complex conditions. Both simple and complex conditions can be enclosed within any number of paired parentheses; the parentheses do not change whether the condition is simple or complex.

### Simple Conditions

There are five simple conditions:

- Class condition
- Condition-name condition
- Relation condition
- Sign condition
- Switch-status condition

A simple condition has a truth value of either true or false.

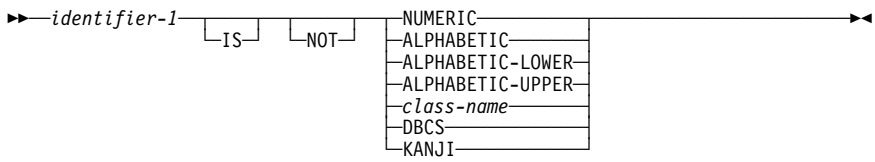
## Conditional Expressions

## Class Condition

The class condition determines whether the content of a data item is alphabetic, alphabetic-lower, alphabetic-upper, numeric, or contains only the characters in the set of characters specified by the CLASS clause as defined in the SPECIAL-NAMES paragraph of the Environment Division.

x The class condition determines whether the contents of a data item are DBCS or
x KANJI.

**Format**




**identifier-1**

Must reference a data item whose usage is explicitly or implicitly DISPLAY.

x Can reference a data item whose usage is explicitly or implicitly DISPLAY-1. If identifier-1 is a function-identifier, it must reference an alphanumeric function.

**NOT**

When used, NOT and the next key word define the class test to be executed for truth value. For example, NOT NUMERIC is a truth test for determining that an identifier is nonnumeric.

**NUMERIC**

Identifier consists entirely of the characters 0 through 9, with or without an operational sign.

If its PICTURE does not contain an operational sign, the identifier being tested is determined to be numeric only if the contents are numeric and an operational sign is not present.

If its PICTURE does contain an operational sign, the identifier being tested is determined to be numeric only if the item is an elementary item, the contents are numeric, and a valid operational sign is present.

The NUMERIC test cannot be used with an identifier described as alphabetic or as a group item that contains one or more signed elementary items.

For numeric data items, the identifier being tested can be described as USAGE
x DISPLAY, (or as IBM extensions) USAGE COMPUTATIONAL-3, or USAGE
x PACKED-DECIMAL.

**ALPHABETIC**

Identifier consists entirely of any combination of the lowercase or uppercase alphabetic characters A through Z and the space.

The ALPHABETIC test cannot be used with an identifier described as numeric.

**ALPHABETIC-LOWER**
Identifier consists entirely of any combination of the lowercase alphabetic characters a through z and the space.

The ALPHABETIC-LOWER test cannot be used with an identifier described as numeric.

**ALPHABETIC-UPPER**
Identifier consists entirely of any combination of the uppercase alphabetic characters A through Z and the space.

The ALPHABETIC-UPPER test cannot be used with an identifier described as numeric.

**class-name**
Identifier consists entirely of the characters listed in the definition of class-name in the SPECIAL-NAMES paragraph.

The class-name test must not be used with an identifier described as numeric.

x **DBCS**

x MVS/VM Under MVS and VM, the identifier consists entirely of DBCS characters.
x For DBCS data items, the identifier being tested must be described explicitly or
x implicitly as USAGE DISPLAY-1. Each byte of the DBCS identifier being tested
x can contain characters that range in value from X'00' through X'FF'.

x WRKSTN Under AIX, OS/2, and Windows, the identifier contains DBCS characters
x that correspond to valid MVS DBCS characters.

x For all platforms, range check is performed on the data portion of the item for valid
x character representation. The valid range is X'41' through X'FE' for both bytes
x of each DBCS character and X'4040' for the DBCS blank.

x **KANJI**

x MVS/VM Under MVS and VM, the identifier consists entirely of DBCS characters.
x For KANJI data items, the identifier being tested must be described explicitly or
x implicitly as USAGE DISPLAY-1. Each byte of the DBCS identifier being tested
x can contain characters that range in value from X'00' through X'FF'.

x WRKSTN Under AIX, OS/2, and Windows the identifier contains DBCS characters
x that correspond to valid MVS DBCS characters.

x For all platforms, a range check is performed on the data portion of the item for
x valid character representation. The valid range is from X'41' through X'7F' for
x the first byte, from X'41' through X'FE' for the second byte, and X'4040' for the
x DBCS blank.

The class test is not valid for items defined as USAGE IS INDEX, as these items do not belong to any class or category.

x The class test is not valid for items defined as USAGE IS POINTER or USAGE IS
x PROCEDURE-POINTER, as these items do not belong to any class or category.

x The class condition cannot be used for external floating-point (USAGE DISPLAY) or
x internal floating-point (USAGE COMP-1 and USAGE COMP-2) items.

Table 17 shows valid forms of the class test.

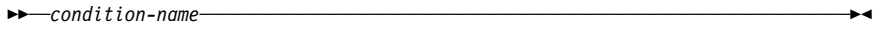
*Table 17. Valid Forms of the Class Test for Different Types of Identifiers*

| Type of Identifier | Valid Forms of the Class Test | |
|---|---|---|
| Alphabetic | ALPHABETIC<br>ALPHABETIC-LOWER<br>ALPHABETIC-UPPER<br>class-name | NOT ALPHABETIC<br>NOT ALPHABETIC-LOWER<br>NOT ALPHABETIC-UPPER<br>NOT class-name |
| Alphanumeric,<br>Alphanumeric-edited, or<br>Numeric-edited | ALPHABETIC<br>ALPHABETIC-LOWER<br>ALPHABETIC-UPPER<br>NUMERIC<br>class-name | NOT ALPHABETIC<br>NOT ALPHABETIC-LOWER<br>NOT ALPHABETIC-UPPER<br>NOT NUMERIC<br>NOT class-name |
| External-Decimal<br>x or Internal-Decimal | NUMERIC | NOT NUMERIC |
| x DBCS<br>x | DBCS<br>KANJI | NOT DBCS<br>NOT KANJI |

## Condition-Name Condition

A condition-name condition tests a conditional variable to determine whether its value is equal to any value(s) associated with the condition-name.

**Format**




A condition-name is used in conditions as an abbreviation for the relation condition. The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

If the condition-name has been associated with a range of values (or with several ranges of values), the conditional variable is tested to determine whether or not its value falls within the range(s), including the end values. The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

x Condition-names with DBCS and floating-point values are allowed.

The following example illustrates the use of conditional variables and condition-names:

```
01  AGE-GROUP           PIC  99.
    88  INFANT          VALUE 0.
    88  BABY            VALUE 1, 2.
    88  CHILD           VALUE 3 THRU 12.
    88  TEEN-AGER       VALUE 13 THRU 19.
```

AGE-GROUP is the conditional variable; INFANT, BABY, CHILD, and TEEN-AGER are condition-names. For individual records in the file, only one of the values specified in the condition-name entries can be present.

The following IF statements can be added to the above example to determine the age group of a specific record:

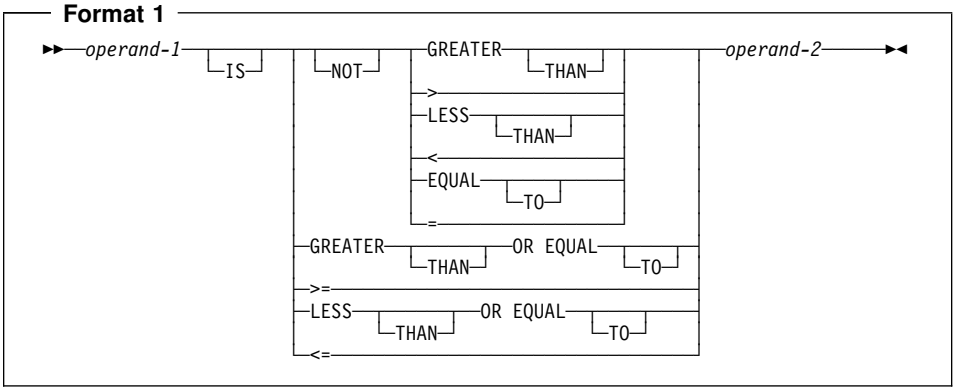
```
IF INFANT...            (Tests for value 0)
IF BABY...              (Tests for values 1, 2)
IF CHILD...             (Tests for values 3 through 12)
IF TEEN-AGER...         (Tests for values 13 through 19)
```

Depending on the evaluation of the condition-name condition, alternative paths of execution are taken by the object program.

## Relation Condition

A relation condition compares two operands, either of which can be an identifier, literal, arithmetic expression, or index-name. A nonnumeric literal can be enclosed in parentheses within a relation condition.

**Format 1**




**operand-1**
The subject of the relation condition. Can be an identifier, literal, function-identifier, arithmetic expression, or index-name.

**operand-2**
The object of the relation condition. Can be an identifier, literal, function-identifier, arithmetic expression, or index-name.

## Conditional Expressions

The relation condition must contain at least one reference to an identifier.

The relational operator specifies the type of comparison to be made. Table 18 shows relational operators and their meanings. Each relational operator must be preceded and followed by a space. The relational operators >= and <= must not have a space between them.

*Table 18. Relational Operators and Their Meanings*

| Relational Operator | Can Be Written | Meaning |
|---|---|---|
| IS GREATER THAN | IS **>** | Greater than |
| IS NOT GREATER THAN | IS NOT **>** | Not greater than |
| IS LESS THAN | IS **<** | Less than |
| IS NOT LESS THAN | IS NOT **<** | Not less than |
| IS EQUAL TO | IS **=** | Equal to |
| IS NOT EQUAL TO | IS NOT **=** | Not equal to |
| IS GREATER THAN OR EQUAL TO | IS **>=** | Is greater than or equal to |
| IS LESS THAN OR EQUAL TO | IS **<=** | Is less than or equal to |

### DBCS Items

MVS/VM Under MVS and VM, DBCS data items and literals can be used with all relational operators. Comparisons are based on the binary collating sequence of the hexadecimal values of the DBCS characters. If the DBCS items are not the same length, the smaller item is padded on the right with DBCS spaces.

WRKSTN Under AIX, OS/2, and Windows, comparisons of DBCS data items and literals are based on the collating sequence as specified by the locale. For more information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 543.

**Note:** The PROGRAM COLLATING SEQUENCE clause will not be applied in comparisons of DBCS data items and literals.

DBCS items can be compared only with DBCS items.

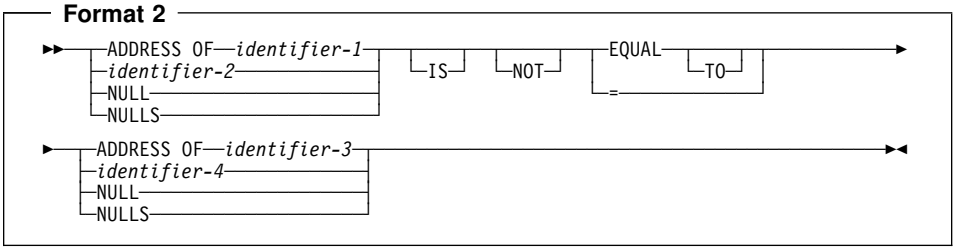
### Pointer Data Items

Only EQUAL and NOT EQUAL are allowed as relational operators when specifying pointer data items. Pointer data items are items defined explicitly as USAGE IS POINTER, or are ADDRESS OF special registers, which are implicitly defined as USAGE IS POINTER.

The operands are equal if the two addresses used in the comparison would both result in the same storage location.

This relation condition is allowed in IF, PERFORM, EVALUATE, and SEARCH Format 1 statements. It is not allowed in SEARCH Format 2 (SEARCH ALL) statements, because there is no meaningful ordering that can be applied to pointer data items.

**Format 2**




**identifier-1**
**identifier-3**
Can specify any level item defined in the Linkage Section, except 66 and 88.

**identifier-2**
**identifier-4**
Must be described as USAGE IS POINTER.

**NULL(S)**
As in this syntax diagram, can be used only if the other operand is defined as USAGE IS POINTER. That is, NULL=NULL is not allowed.

Table 19 summarizes the permissible comparisons for USAGE IS POINTER, NULL, and ADDRESS OF.

*Table 19. Permissible Comparisons for USAGE IS POINTER, NULL, and ADDRESS OF*

| | Second Operand | | |
|---|---|---|---|
| **First Operand** | USAGE IS POINTER | ADDRESS OF | NULL/NULLS |
| USAGE IS POINTER | Yes | Yes | Yes |
| ADDRESS OF | Yes | Yes | Yes |
| NULL/NULLS | Yes | Yes | No |

**Note:**
YES = Comparison allowed only for EQUAL, NOT EQUAL
NO = No comparison allowed

## Procedure-Pointer Data Items

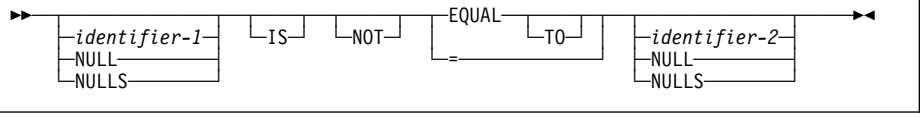
Only EQUAL and NOT EQUAL are allowed as relational operators when specifying procedure-pointer data items. Procedure-pointer data items are items defined explicitly as USAGE IS PROCEDURE-POINTER.

The operands are equal if the two addresses used in the comparison would both result in the same storage location.

## Conditional Expressions

This relation condition is allowed in IF, PERFORM, EVALUATE, and SEARCH Format 1 statements. It is not allowed in SEARCH Format 2 (SEARCH ALL) statements, because there is no meaningful ordering that can be applied to procedure-pointer data items.

**Format 3**



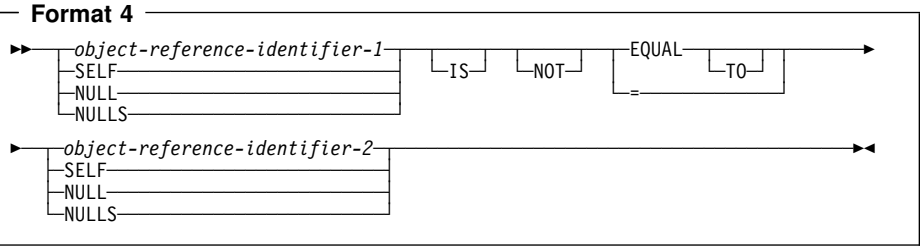

**identifier-1**
**identifier-2**
Must be described as USAGE IS PROCEDURE-POINTER.

**NULL(S)**
As in this syntax diagram, can be used only if the other operand is defined as USAGE IS PROCEDURE-POINTER. That is, NULL=NULL is not allowed.

### Object Reference Data Items

A data item of USAGE OBJECT REFERENCE can be compared for equality or inequality with another data item of USAGE OBJECT REFERENCE or with NULL, NULLS, or SELF. (A comparison with SELF is only allowed in a method.) Two object-references compare equal only if the data items identify the same object.

**Format 4**




## Comparison of Numeric and Nonnumeric Operands

### Comparing Numeric Operands

The algebraic values of numeric operands are compared.

- The length (number of digits) of the operands is not significant.
- Unsigned numeric operands are considered positive.
- Zero is considered to be a unique value, regardless of sign.
- Comparison of numeric operands is permitted, regardless of the type of USAGE specified for each.

Table 20 summarizes permissible comparisons with **numeric** operands.

The symbols used in Table 20 and Table 21 are as follows:

NN = Comparison for nonnumeric operands
NU = Comparison for numeric operands
Blank = Comparison is not allowed.

*Table 20. Permissible Comparisons with Numeric Second Operands*

| | | Second Operand | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| x | **First Operand** | **ZR** | **NL** | **ED** | **BI** | **AE** | **ID** | **IFP** | **EFP** | **FPL** |
| | **Nonnumeric Operand** | | | | | | | | | |
| x | **Group (GR)** | NN | NN[1] | NN[1] | | | | | NN | |
| x | **Alphabetic (AL)** | NN | NN[1] | NN[1] | | | | | NN | |
| x | **Alphanumeric (AN)** | NN | NN[1] | NN[1] | | | | | NN | |
| x | **Alphanumeric-edited (ANE)** | NN | NN[1] | NN[1] | | | | | NN | |
| x | **Numeric-Edited (NE)** | NN | NN[1] | NN[1] | | | | | NN | |
| x | **Figurative Constant (FC[2])** | | | NN[1] | | | | | NN | |
| x | **Nonnumeric Literal (NNL)** | | | NN[1] | | | | | NN | |
| | **Numeric Operand** | | | | | | | | | |
| x<br>x<br>x | **Figurative Constant ZERO (ZR)** | | | NU | NU | NU | NU | NU | NU | |
| x | **Numeric Literal (NL)** | | | NU | NU | NU | NU | NU | NU | |
| x | **External Decimal (ED)** | NU | NU | NU | NU | NU | NU | NU | NU | NU |
| x | **Binary (BI)** | NU | NU | NU | NU | NU | NU | NU | NU | NU |
| x | **Arithmetic Expression (AE)** | NU | NU | NU | NU | NU | NU | NU | NU | NU |
| x | **Internal Decimal (ID)** | NU | NU | NU | NU | NU | NU | NU | NU | NU |
| x | **Internal Floating-point (IFP)** | NU | NU | NU | NU | NU | NU | NU | NU | NU |
| x | **External Floating-Point (EFP)** | NU | NU | NU | NU | NU | NU | NU | NU | NU |
| x | **Floating-point Literal (FPL)** | | | NU | NU | NU | NU | NU | NU | |

**Note:**

1 Integer item only.
2 Includes all figurative constants except ZERO.

## Conditional Expressions

### Comparing Nonnumeric Operands

Comparisons of nonnumeric operands are made with respect to the collating sequence of the character set in use.

- For the EBCDIC character set, the EBCDIC collating sequence is used.
- For the ASCII character set, the ASCII collating sequence is used. (See Appendix B, “EBCDIC and ASCII Collating Sequences” on page 522.)
- WRKSTN Under AIX, OS/2, and Windows if the collating sequence specified is NATIVE (explicitly or by default), the comparisons of characters are based on the collating sequence indicated by the locale setting. For more information on locale, see Appendix F, “Locale Considerations (Workstation Only)” on page 543.
- When the PROGRAM COLLATING SEQUENCE clause is specified in the OBJECT-COMPUTER paragraph, the collating sequence associated with the alphabet-name clause in the SPECIAL-NAMES paragraph is used.

The size of each operand is the total number of characters in that operand; the size affects the result of the comparison. There are two cases to consider:

**Operands of Equal Size**

Characters in corresponding positions of the two operands are compared, beginning with the leftmost character and continuing through the rightmost character.

If all pairs of characters through the last pair test as equal, the operands are considered as equal.

If a pair of unequal characters is encountered, the characters are tested to determine their relative positions in the collating sequence. The operand containing the character higher in the sequence is considered the greater operand.

**Operands of Unequal Size**

If the operands are of unequal size, the comparison is made as though the shorter operand were extended to the right with enough spaces to make the operands equal in size.

Table 21 on page 235 summarizes permissible comparisons with **nonnumeric** operands.

*Table 21. Permissible Comparisons with Nonnumeric Second Operands*

| | Second Operand | | | | | | |
|---|---|---|---|---|---|---|---|
| **First Operand** | **GR** | **AL** | **AN** | **ANE** | **NE** | **FC**[2] | **NNL** |
| **Nonnumeric Operand** | | | | | | | |
| **Group (GR)** | NN | NN | NN | NN | NN | NN | NN |
| **Alphabetic (AL)** | NN | NN | NN | NN | NN | NN | NN |
| **Alphanumeric (AN)** | NN | NN | NN | NN | NN | NN | NN |
| **Alphanumeric-edited (ANE)** | NN | NN | NN | NN | NN | NN | NN |
| **Numeric-Edited (NE)** | NN | NN | NN | NN | NN | NN | NN |
| **Figurative Constant (FC**[2]**)** | NN | NN | NN | NN | NN | | |
| **Nonnumeric Literal (NNL)** | NN | NN | NN | NN | NN | | |
| **Numeric Operand** | | | | | | | |
| **Figurative Constant ZERO (ZR)** | NN | NN | NN | NN | NN | | |
| **Numeric Literal (NL)** | NN[1] | NN[1] | NN[1] | NN[1] | NN[1] | | |
| **External Decimal (ED)** | NN[1] | NN[1] | NN[1] | NN[1] | NN[1] | NN[1] | NN[1] |
| **Binary (BI)** | | | | | | | |
| **Arithmetic Expression (AE)** | | | | | | | |
| **Internal Decimal (ID)** | | | | | | | |
| **Internal Floating-point (IFP)** | | | | | | | |
| **External Floating-point (EFP)** | NN | NN | NN | NN | NN | NN | NN |
| **Floating-point Literal (FPL)** | | | | | | | |

**Note:**

1 Integer item only.
2 Includes all figurative constants except ZERO.

## Conditional Expressions

### Comparing Numeric and Nonnumeric Operands

The nonnumeric comparison rules, discussed above, apply. In addition, when numeric and nonnumeric operands are being compared, their USAGE must be the same. In such comparisons:

- The numeric operand must be described as an integer literal or data item.
- Non-integer literals and data items must not be compared with nonnumeric operands.
- x External floating-point items can be compared with nonnumeric operands.

If either of the operands is a group item, the nonnumeric comparison rules, discussed above, apply. In addition to those rules:

- If the nonnumeric operand is a **literal or an elementary data item**, the numeric operand is treated as though it were moved to an alphanumeric elementary data item of the same size, and the contents of this alphanumeric data item were then compared with the nonnumeric operand.
- If the nonnumeric operand is a **group item**, the numeric operand is treated as though it were moved to a group item of the same size, and the contents of this group item were compared then with the nonnumeric operand.

  See “MOVE Statement” on page 333.

### Comparing Index-Names and Index Data Items

Comparisons involving index-names and/or index data items conform to the following rules:

- The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.
- In the comparison of an index-name with a data item (other than an index data item), or in the comparison of an index-name with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.
- x In the comparison of an index-name with an arithmetic expression, the occurrence x number that corresponds to the value of the index-name is compared with the x arithmetic expression.

  x Since an integer function can be used wherever an arithmetic expression can be x used, this extension allows you to compare an index-name to an integer or numeric x function.
- In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion. Results of any other comparison involving an index data item are undefined.

Table 22 shows valid comparisons for index-names and index data items.

*Table 22. Comparisons for Index-Names and Index Data Items*

| Operands Compared | Index-Name | Index Data Item | Data-Name (Numeric Integer Only) | Literal (Numeric Integer Only) | Arithmetic Expression |
|---|---|---|---|---|---|
| **Index-Name** | Compare occurrence number | Compare without conversion | Compare occurrence number with data-name | Compare occurrence number with literal | Compare occurrence number with arithmetic expression |
| **Index Data Item** | Compare without conversion | Compare without conversion | Illegal | Illegal | Illegal |

## Comparison of DBCS Operands

The rules for comparing DBCS operands are the same as those for the comparison of nonnumeric operands.

MVS/VM Under MVS and VM, the comparison is based on a binary collating sequence of the hexadecimal values of the DBCS characters.

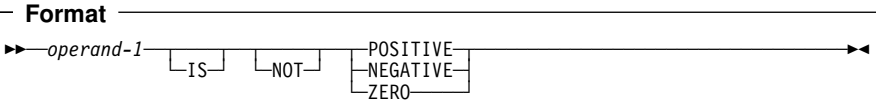
WRKSTN Under AIX, OS/2, and Windows if the collating sequence specified is NATIVE (explicitly or by default), the comparisons of characters are based on the collating sequence indicated by the locale setting. For more information on locale, see Appendix F, “Locale Considerations (Workstation Only)” on page 543.

**Note:** The PROGRAM COLLATING SEQUENCE clause will not be applied to comparisons of DBCS operands.

## Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand is greater than, less than, or equal to zero.

**Format**




**operand**

Must be defined as a numeric identifier, or it must be defined as an arithmetic expression that contains at least one reference to a variable. The operand can be defined as a floating-point identifier.

## Conditional Expressions

The operand is:

- POSITIVE if its value is greater than zero
- NEGATIVE if its value is less than zero
- ZERO if its value is equal to zero

An unsigned operand is either POSITIVE or ZERO.

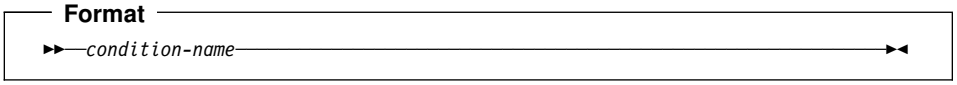
**NOT**
One algebraic test is executed for the truth value of the sign condition. For example, NOT ZERO is regarded as true when the operand tested is positive or negative in value.

x MVS/VM Under MVS and VM, if you are using the NUMPROC compiler option,
x the results of the sign condition test can be affected. For details, see the *IBM*
x *COBOL for MVS & VM Programming Guide*.

## Switch-Status Condition

The switch-status condition determines the on or off status of an UPSI switch.

**Format**




**condition-name**
Must be defined in the SPECIAL-NAMES paragraph as associated with the ON or OFF value of an UPSI switch. (See "SPECIAL-NAMES Paragraph" on page 88.)

The switch-status condition tests the value associated with the condition-name. (The value associated with the condition-name is considered to be alphanumeric.) The result of the test is true if the UPSI switch is set to the value (0 or 1) corresponding to condition-name. See "UPSI" in *IBM COBOL Programming Guide* for your platform.

## Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions, and/or complex conditions with logical operators, or negating these conditions with logical negation.

Each logical operator must be preceded and followed by a space. The following chart shows the logical operators and their meanings.

*Table 23 (Page 1 of 2). Logical Operators and Their Meanings*

| Logical Operator | Name | Meaning |
|---|---|---|
| AND | Logical conjunction | The truth value is **true** when both conditions are true. |

*Table 23 (Page 2 of 2). Logical Operators and Their Meanings*

| Logical Operator | Name | Meaning |
|---|---|---|
| OR | Logical inclusive OR | The truth value is **true** when either or both conditions are true. |
| NOT | Logical negation | Reversal of truth value (the truth value is **true** if the condition is false). |

Unless modified by parentheses, the following precedence rules (from highest to lowest) apply:

1. Arithmetic operations
2. Simple conditions
3. NOT
4. AND
5. OR

The truth value of a complex condition (whether parenthesized or not) is the truth value that results from the interaction of all the stated logical operators on either of the following:

- The individual truth values of simple conditions
- The intermediate truth values of conditions logically combined or logically negated.

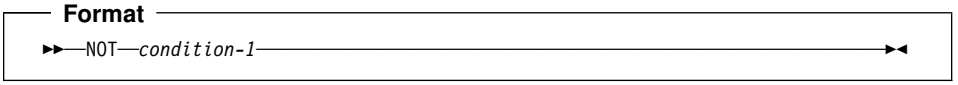
A complex condition can be either of the following:

- A negated simple condition
- A combined condition (which can be negated)

## Negated Simple Conditions

A simple condition is negated through the use of the logical operator NOT.

**Format**




The negated simple condition gives the opposite truth value of the simple condition. That is, if the truth value of the simple condition is true, then the truth value of that same negated simple condition is false, and vice versa.

Placing a negated simple condition within parentheses does not change its truth value. That is, the following two statements are equivalent:

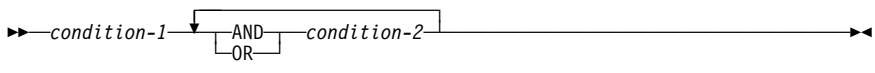
```
NOT A IS EQUAL TO B.

NOT (A IS EQUAL TO B).
```

## Conditional Expressions

## Combined Conditions

Two or more conditions can be logically connected to form a combined condition.

**Format**




The condition to be combined can be any of the following:

- A simple-condition
- A negated simple-condition
- A combined condition
- A negated combined condition (that is, the NOT logical operator followed by a combined condition enclosed in parentheses)
- Combinations of the preceding conditions that are specified according to the rules in the following table.

*Table 24. Combined Conditions—Permissible Element Sequences*

| Combined condition element | Left most | When not leftmost, can be immediately preceded by: | Right most | When not rightmost, can be immediately followed by: |
|---|---|---|---|---|
| simple- condition | Yes | OR NOT AND ( | Yes | OR AND ) |
| OR AND | No | simple-condition ) | No | simple-condition NOT ( |
| NOT | Yes | OR AND ( | No | simple-condition ( |
| ( | Yes | OR NOT AND ( | No | simple-condition NOT ( |
| ) | No | simple-condition ) | Yes | OR AND ) |

Parentheses are never needed when either ANDs or ORs (but not both) are used exclusively in one combined condition. However, parentheses can be needed to modify the implicit precedence rules to maintain the correct logical relation of operators and operands.

There must be a one-to-one correspondence between left and right parentheses, with each left parenthesis to the left of its corresponding right parenthesis.

Table 25 on page 241 illustrates the relationships between logical operators and conditions C1 and C2.

*Table 25. Logical Operators and Evaluation Results of Combined Conditions*

| Value for C1 | Value for C2 | C1 AND C2 | C1 OR C2 | NOT (C2 AND C2) | NOT C1 AND C2 | NOT (C1 OR C2) | NOT C1 OR C2 |
|---|---|---|---|---|---|---|---|
| True | True | True | True | False | False | False | True |
| False | True | False | True | True | True | False | True |
| True | False | False | True | True | False | False | False |
| False | False | False | False | True | False | True | True |

## Order of Evaluation of Conditions

Parentheses, both explicit and implicit, define the level of inclusiveness within a complex condition. Two or more conditions connected by only the logical operators AND or OR at the same level of inclusiveness establish a hierarchical level within a complex condition. An entire complex condition, therefore, is a nested structure of hierarchical levels with the entire complex condition 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 condition. 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.

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. For example:

```
NOT A IS GREATER THAN B OR A + B IS EQUAL TO C AND D IS POSITIVE
```

is evaluated as if parenthesized as follows:

```
(NOT (A IS GREATER THAN B)) OR
(((A + B) IS EQUAL TO C) AND (D IS POSITIVE))
```

## Order of Evaluation:

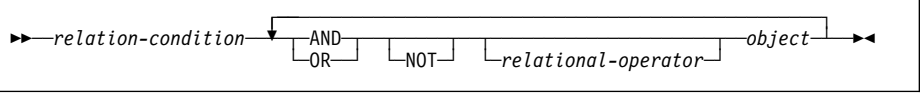
1. `(NOT (A IS GREATER THAN B))` is evaluated, giving some intermediate truth value, **t1**. If **t1** is true, the combined condition is true, and no further evaluation takes place. If **t1** is false, evaluation continues as follows.
2. `(A + B)` is evaluated, giving some intermediate result, **x**.
3. (**x** `IS EQUAL TO C`) is evaluated, giving some intermediate truth value, **t2**. If **t2** is false, the combined condition is false, and no further evaluation takes place. If **t2** is true, the evaluation continues as follows.
4. `(D IS POSITIVE)` is evaluated, giving some intermediate truth value, **t3**. If **t3** is false, the combined condition is false. If **t3** is true, the combined condition is true.

## Abbreviated Combined Relation Conditions

When relation-conditions are written consecutively, any relation-condition after the first can be abbreviated in one of two ways:

- Omission of the subject
- Omission of the subject and relational operator.

**Format**




In any consecutive sequence of relation-conditions, both forms of abbreviation can be specified. The abbreviated condition is evaluated as if:

1. The last stated subject is the missing subject.
2. The last stated relational operator is the missing relational operator.

The resulting combined condition must comply with the rules for element sequence in combined conditions, as shown in Table 24 on page 240.

If the word immediately following NOT is GREATER THAN, >, LESS THAN, <, EQUAL TO, and =, then the NOT participates as part of the relational operator.

NOT in any other position is considered a logical operator (and thus results in a negated relation-condition).

### Using Parentheses

You can use parentheses in combined relation conditions to specify an intended order of evaluation. Using parentheses can also help you to improve the readability of conditional expressions.

The following rules govern the use of parentheses in abbreviated combined relation conditions:

1. Parentheses can be used to change the order of evaluation of the logical operators AND and OR.
2. The word NOT participates as part of the relational operator when it is immediately followed by GREATER THAN, >, LESS THAN, <, EQUAL TO, and =.
3. NOT in any other position is considered a logical operator and thus results in a negated relation-condition. If you use NOT as a logical operator, only the relation condition immediately following the NOT is negated; the negation is not propagated through the abbreviated combined relation condition along with the subject and relational operator.
4. The logical NOT operator can appear within a parenthetical expression that immediately follows a relational operator.
5. When a left parenthesis appears immediately after the relational operator, the relational operator is distributed to all objects enclosed in the parentheses. In the case

of a “distributed” relational operator, the subject and relational operator remain current after the right parenthesis which ends the distribution. The following three restrictions apply to cases where the relational operator is distributed throughout the expression:

a. A simple condition cannot appear within the scope of the distribution.
b. Another relational operator cannot appear within the scope of the distribution.
c. The logical operator NOT cannot appear immediately after the left parenthesis, which defines the scope of the distribution.

6. Evaluation proceeds from the least to the most inclusive condition.
7. There must be a one-to-one correspondence between left and right parentheses, with each left parenthesis to the left of its corresponding right parenthesis. If the parentheses are unbalanced, the compiler inserts a parenthesis and issues an E-level message. Note, however, that if the compiler-inserted parenthesis results in the truncation of the expression, you will receive an S-level diagnostic message.
8. The last stated subject is inserted in place of the missing subject.
9. The last stated relational operator is inserted in place of the missing relational operator.
10. Insertion of the omitted subject and/or relational operator ends when:
    a. Another simple condition is encountered,
    b. A condition-name is encountered,
    c. A right parenthesis is encountered that matches a left parenthesis that appears to the left of the subject.
11. In any consecutive sequence of relation conditions, you can use both abbreviated relation conditions that contain parentheses and those that don't.
12. Consecutive logical NOT operators cancel each other and result in an S-level message. Note, however, that an abbreviated combined relation condition can contain two consecutive NOT operators when the second NOT is part of a relational operator. For example, you can abbreviate the first condition as the second condition listed below.

```
A = B and not A not = C
A = B and not not = C
```

The following chart summarizes the rules for forming an abbreviated combined relation condition.

## Conditional Expressions

*Table 26. Abbreviated Combined Conditions—Permissible Element Sequences*

| Combined Condition Element | Left most | When not leftmost, can be immediately preceded by: | Right most | When not rightmost, can be immediately followed by: |
|---|---|---|---|---|
| Subject | Yes | NOT ( | No | Relational operator |
| Object | No | Relational operator AND OR NOT ( | Yes | AND OR ) |
| Relational operator | No | Subject AND OR NOT | No | Object ( |
| AND OR | No | Object ) | No | Object Relational operator NOT ( |
| NOT | Yes | AND OR ( | No | Subject Object Relational operator ( |
| ( | Yes | Relational operator AND OR NOT ( | No | Subject Object NOT ( |
| ) | No | Object ) | Yes | AND OR ) |

The following examples illustrate abbreviated combined relation conditions, with and without parentheses, and their unabbreviated equivalents.

*Table 27. Abbreviated Combined Conditions—Unabbreviated Equivalents*

| Abbreviated Combined Relation Condition | Equivalent |
|---|---|
| A = B AND NOT < C OR D | ((A = B) AND (A NOT < C)) OR (A NOT < D) |
| A NOT > B OR C | (A NOT > B) OR (A NOT > C) |
| NOT A = B OR C | (NOT (A = B)) OR (A = C) |
| NOT (A = B OR < C) | NOT ((A = 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)))) |

## Statement Categories

There are four categories of COBOL statements:

- Imperative
- Conditional
- Delimited scope
- Compiler directing.

## Imperative Statements

An **imperative statement** either specifies an unconditional action to be taken by the program, or is a conditional statement terminated by its explicit scope terminator (see "Delimited Scope Statements" on page 247). A series of imperative statements can be specified whenever an imperative statement is allowed. A conditional statement that is terminated by its explicit scope terminator is also classified as an imperative statement (see "Delimited Scope Statements" on page 247). Table 28 lists COBOL imperative statements.

*Table 28 (Page 1 of 2). Imperative Statements*

| | |
|---|---|
| **Arithmetic** | |
| | ADD[1] |
| | COMPUTE[1] |
| | DIVIDE[1] |
| | MULTIPLY[1] |
| | SUBTRACT[1] |
| **Data Movement** | |
| | ACCEPT (DATE,DAY,DAY-OF-WEEK,TIME) |
| | INITIALIZE |
| | INSPECT |
| | MOVE |
| | SET |
| | STRING[2] |
| | UNSTRING[2] |
| **Ending** | |
| | STOP RUN |
| | EXIT PROGRAM |
| x | EXIT METHOD |
| x | GOBACK |
| **Input-Output** | |
| | ACCEPT identifier |
| | CLOSE |
| | DELETE[3] |
| | DISPLAY |
| | OPEN |
| | READ[4] |
| | REWRITE[3] |
| | START[3] |
| | STOP literal |
| | WRITE[5] |

## Statement Categories

*Table 28 (Page 2 of 2). Imperative Statements*

| | |
|---|---|
| **Ordering** | |
| | MERGE |
| | RELEASE |
| | RETURN[6] |
| | SORT |
| **Procedure Branching** | |
| | ALTER |
| | EXIT |
| | GO TO |
| | PERFORM |
| **Program or Method Linkage** | |
| | CALL[7] |
| | CANCEL |
| | INVOKE |
| **Table Handling** | |
| | SET |

**Note:**

1 Without the ON SIZE ERROR and/or the NOT ON SIZE ERROR phrase.
2 Without the ON OVERFLOW and/or the NOT ON OVERFLOW phrase.
3 Without the INVALID KEY and/or the NOT INVALID KEY phrase.
4 Without the AT END, NOT AT END, INVALID KEY, and/or NOT INVALID KEY phrases.
5 Without the INVALID KEY, NOT INVALID KEY, END-OF-PAGE, and/or NOT END-OF-PAGE phrases.
6 Without the AT END and/or NOT AT END phrase.
7 Without the ON OVERFLOW phrase, and without the ON EXCEPTION and/or NOT ON EXCEPTION phrase.

## Conditional Statements

A **conditional statement** specifies that the truth value of a condition is to be determined, and that the subsequent action of the object program is dependent on this truth value. (See “Conditional Expressions” on page 225.) Table 29 lists COBOL statements that become conditional when a **condition** (for example, ON SIZE ERROR or ON OVERFLOW) is included, and when the statement is not terminated by its explicit scope terminator.

*Table 29 (Page 1 of 2). Conditional Statements*

| |
|---|
| **Arithmetic** |
| ADD...ON SIZE ERROR |
| ADD...NOT ON SIZE ERROR |
| COMPUTE...ON SIZE ERROR |
| COMPUTE...NOT ON SIZE ERROR |
| DIVIDE...ON SIZE ERROR |
| DIVIDE...NOT ON SIZE ERROR |
| MULTIPLY...ON SIZE ERROR |
| MULTIPLY...NOT ON SIZE ERROR |
| SUBTRACT...ON SIZE ERROR |
| SUBTRACT...NOT ON SIZE ERROR |

*Table 29 (Page 2 of 2). Conditional Statements*

| | |
|---|---|
| **Data Movement** | |
| | STRING...ON OVERFLOW |
| | STRING...NOT ON OVERFLOW |
| | UNSTRING...ON OVERFLOW |
| | UNSTRING...NOT ON OVERFLOW |
| **Decision** | |
| | IF |
| | EVALUATE |
| **Input-Output** | |
| | DELETE...INVALID KEY |
| | DELETE...NOT INVALID KEY |
| | READ...AT END |
| | READ...NOT AT END |
| | READ...INVALID KEY |
| | READ...NOT INVALID KEY |
| | REWRITE...INVALID KEY |
| | REWRITE...NOT INVALID KEY |
| | START...INVALID KEY |
| | START...NOT INVALID KEY |
| | WRITE...AT END-OF-PAGE |
| | WRITE...NOT AT END-OF-PAGE |
| | WRITE...INVALID KEY |
| | WRITE...NOT INVALID KEY |
| **Ordering** | |
| | RETURN...AT END |
| | RETURN...NOT AT END |
| x **Program or Method Linkage** | |
| | CALL...ON OVERFLOW |
| | CALL...ON EXCEPTION |
| | CALL...NOT ON EXCEPTION |
| x | INVOKE...ON EXCEPTION |
| x | INVOKE...NOT ON EXCEPTION |
| **Table Handling** | |
| | SEARCH |

## Delimited Scope Statements

In general, a DELIMITED SCOPE statement uses an explicit scope terminator to turn a conditional statement into an imperative statement; the resulting imperative statement can then be nested. Explicit scope terminators can also be used, however, to terminate the scope of an imperative statement. Explicit scope terminators are provided for all COBOL verbs that can have conditional phrases.

Unless explicitly specified otherwise, a delimited scope statement can be specified wherever an imperative statement is allowed by the rules of the language.

## Statement Categories

### Explicit Scope Terminators

An EXPLICIT SCOPE TERMINATOR marks the end of certain Procedure Division statements. A conditional statement that is delimited by its explicit scope terminator is considered an imperative statement and must follow the rules for imperative statements.

The following are explicit scope terminators:

```
END-ADD              END-READ
END-CALL             END-RETURN
END-COMPUTE          END-REWRITE
END-DELETE           END-SEARCH
END-DIVIDE           END-START
END-EVALUATE         END-STRING
END-IF               END-SUBTRACT
x END-INVOKE         END-UNSTRING
END-MULTIPLY         END-WRITE
END-PERFORM
```

### Implicit Scope Terminators

At the end of any sentence, an IMPLICIT SCOPE TERMINATOR is a separator period that terminates the scope of all previous statements not yet terminated.

An unterminated conditional statement cannot be contained by another statement. However, a scope terminator will be assumed just prior to the next phrase of the containing statement.

**Note:** Except for nesting conditional statements within IF statements, nested statements must be imperative statements, and must follow the rules for imperative statements. You should not nest conditional statements.

### Compiler-Directing Statements

Statements that direct the compiler to take a specified action are discussed in "Compiler-Directing Statement" on page 482.

## Statement Operations

COBOL statements perform the following types of operations:

- Arithmetic
- Data manipulation
- Input/output
- Procedure branching

There are several phrases common to arithmetic and data manipulation statements, such as:

- CORRESPONDING Phrase
- GIVING Phrase
- ROUNDED Phrase
- SIZE ERROR Phrases

## CORRESPONDING Phrase

The CORRESPONDING phrase (CORR) allows ADD, SUBTRACT, and MOVE operations to be performed on elementary data items of the same name if the group items to which they belong are specified.

Both identifiers following the key word CORRESPONDING must name group items. In this discussion, these identifiers are referred to as identifier-1 and identifier-2.

A pair of data items (subordinate items), one from identifier-1 and one from identifier-2, correspond if the following conditions are true:

- In an ADD or SUBTRACT statement, both of the data items are elementary numeric data items. Other data items are ignored.
- In a MOVE statement, at least one of the data items is an elementary item, and the move is permitted by the move rules.
- The two subordinate items have the same name and the same qualifiers up to but not including identifier-1 and identifier-2.
- The subordinate items are not identified by the key word FILLER.
- Neither identifier-1 nor identifier-2 is described as a level 66, 77, or 88 item, nor is either described as a USAGE IS INDEX item. Neither identifier-1 nor identifier-2 can be reference-modified.
- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, USAGE IS INDEX, and USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE clause in their descriptions.

  However, identifier-1 and identifier-2 themselves can contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.
- Neither identifier-1 nor identifier-2 is described as a USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE
- identifier-1 and/or identifier-2 can be subordinate to a FILLER item.

For example, if two data hierarchies are defined as follows:

```
05  ITEM-1 OCCURS 6.
  10  ITEM-A PIC S9(3).
  10  ITEM-B PIC +99.9.
  10  ITEM-C PIC X(4).
  10  ITEM-D REDEFINES ITEM-C PIC 9(4).
  10  ITEM-E USAGE COMP-1.
  10  ITEM-F USAGE INDEX.
05  ITEM-2.
  10  ITEM-A PIC 99.
  10  ITEM-B PIC +9V9.
  10  ITEM-C PIC A(4).
  10  ITEM-D PIC 9(4).
  10  ITEM-E PIC 9(9) USAGE COMP.
  10  ITEM-F USAGE INDEX.
```

Then, if ADD CORR ITEM-2 TO ITEM-1(X) is specified, ITEM-A and ITEM-A(X), ITEM-B and ITEM-B(X), and ITEM-E and ITEM-E(X) are considered to be corresponding and are added together. ITEM-C and ITEM-C(X) are not included because they are not numeric. ITEM-D and ITEM-D(X) are not included because ITEM-D(X) includes a REDEFINES clause in its data description. ITEM-F and ITEM-F(X) are not included because they are defined as USAGE IS INDEX. Note that ITEM-1 is valid as either identifier-1 or identifier-2.

If any of the individual operations in the ADD CORRESPONDING statement produces a size error condition, imperative-statement-1 in the ON SIZE ERROR phrase is not executed until all of the individual additions are completed.

## GIVING Phrase

The value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it can be a numeric-edited item.

## ROUNDED Phrase

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

When the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless ROUNDED is specified. When ROUNDED is specified, the least significant digit of the resultant identifier is increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps, and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs, relative to the rightmost integer position for which storage is allocated.

x In a floating-point arithmetic operation, the ROUNDED phrase has no effect; the result
x of a floating-point operation is **always** rounded. For more information on floating-point
x arithmetic expressions, see *IBM COBOL Programming Guide* for your platform.

## SIZE ERROR Phrases

A size error condition can occur in three different ways:

- When the absolute value of the result of an arithmetic evaluation, after decimal point alignment, exceeds the largest value that can be contained in the result field
- When division by zero occurs
- In an exponential expression, as indicated in the following table:

*Table 30. Exponentiation Size Error Conditions*

| Size error | Action taken when a SIZE ERROR clause is present | Action taken when a SIZE ERROR clause is not present |
|---|---|---|
| Zero raised to zero power | The SIZE ERROR imperative is executed. | The value returned is 1, and a message is issued. |
| Zero raised to a negative number | The SIZE ERROR imperative is executed. | Program is terminated abnormally. |
| A negative number raised to a fractional power | The SIZE ERROR imperative is executed. | The absolute value of the base is used, and a message is issued. |

The size error condition applies only to final results, not to any intermediate results.

If the resultant identifier is defined with USAGE IS BINARY, COMPUTATIONAL, or
x COMPUTATIONAL-4, the largest value that can be contained in it is the maximum value implied by its associated decimal PICTURE character-string.

If the ROUNDED phrase is specified, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the ON SIZE ERROR phrase is specified.

If the ON SIZE ERROR phrase **is not** specified and a size error condition occurs, truncation rules apply and the value of the affected resultant identifier is computed.

If the ON SIZE ERROR phrase **is** specified and a size error condition occurs, the value of the resultant identifier affected by the size error is not altered—that is, the error results are not placed in the receiving identifier. After completion of the execution of the arithmetic operation, the imperative statement in the ON SIZE ERROR phrase is executed, control is transferred to the end of the arithmetic statement, and the NOT ON SIZE ERROR phrase, if specified, is ignored.

For ADD CORRESPONDING and SUBTRACT CORRESPONDING statements, if an individual arithmetic operation causes a size error condition, the ON SIZE ERROR

imperative statement is not executed until all the individual additions or subtractions have been completed.

If the NOT ON SIZE ERROR phrase has been specified and, after execution of an arithmetic operation, a size error condition does not exist, the NOT ON SIZE ERROR phrase is executed.

When both ON SIZE ERROR and NOT ON SIZE ERROR phrases are specified, and the statement in the phrase that is executed does not contain any explicit transfer of control, then, if necessary, an implicit transfer of control is made after execution of the phrase to the end of the arithmetic statement.

## Arithmetic Statements

The arithmetic statements are used for computations. Individual operations are specified by the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. These operations can be combined symbolically in a formula, using the COMPUTE statement.

## Arithmetic Statement Operands

The data description of operands in an arithmetic statement need not be the same. Throughout the calculation, the compiler performs any necessary data conversion and decimal point alignment.

### Size of Operands

The maximum size of each operand is 18 decimal digits. The **composite of operands** is a hypothetical data item resulting from aligning the operands at the decimal point and then superimposing them on one another. It must not contain more than 18 decimal digits.

For example, assume that each item is defined as follows in the Data Division:

```
A  PICTURE 9(7)V9(5).
B  PICTURE 9(11)V99.
C  PICTURE 9(12)V9(3).
```

If the following statement is executed, the composite of operands consists of 17 decimal digits:

```
ADD A B TO C
```

It has the following implicit description:

```
COMPOSITE-OF-OPERANDS PICTURE 9(12)V9(5).
```

x The composite of operands can be more than 18 digits. For more information, see the
x section on intermediate results in the *IBM COBOL Programming Guide* for your plat-
x form.

In the ADD and SUBTRACT statements, if the composite of operands is 18 digits or less, the compiler ensures that enough places are carried so that no significant digits are lost during execution. The following table shows how the composite of operands is determined for arithmetic statements:

*Table 31. How the Composite of Operands is Determined*

| Statement | Determination of the Composite of Operands |
|---|---|
| SUBTRACT<br>ADD | Superimposing all operands in a given statement (except those following the word GIVING) |
| MULTIPLY | Superimposing all receiving data items |
| DIVIDE | Superimposing all receiving data items, except the REMAINDER data item |
| COMPUTE | Restriction does not apply |

In all arithmetic statements, it is important to define data with enough digits and decimal places to ensure the desired accuracy in the final result.

### Overlapping Operands

When operands in an arithmetic statement share part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

### Multiple Results

When an arithmetic statement has multiple results, execution conceptually proceeds as follows:

- The statement performs all arithmetic operations to find the result to be placed in the receiving items, and stores that result in a temporary location.
- A sequence of statements transfers or combines the value of this temporary result with each single receiving field. The statements are considered to be written in the same left-to-right order as the multiple results are listed.

For example, executing the following statement:

```
ADD A, B, C, TO C, D(C), E.
```

is equivalent to executing the following series of statements:

```
ADD A, B, C GIVING TEMP.
ADD TEMP TO C.
ADD TEMP TO D(C).
ADD TEMP TO E.
```

In the above example, TEMP is a compiler-supplied temporary result field. When the addition operation for D(C) is performed, the subscript C contains the new value of C.

## Data Manipulation Statements

The following COBOL statements move and inspect data: ACCEPT, INITIALIZE, INSPECT, MOVE, READ, RELEASE, RETURN, REWRITE, SET, STRING, UNSTRING, and WRITE.

## Statement Operations

### Overlapping Operands

When the sending and receiving fields of a data manipulation statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

## Input-Output Statements

COBOL input-output statements transfer data to and from files stored on external media, and also control low-volume data that is obtained from or sent to an input/output device.

In COBOL, the unit of file data made available to the program is a record, and you need only be concerned with such records. Provision is automatically made for such operations as the movement of data into buffers and/or internal storage, validity checking, error correction (where feasible), blocking and deblocking, and volume switching procedures.

The description of the file in the Environment Division and Data Division governs which input-output statements are allowed in the Procedure Division. Permissible statements for each type of file organization are shown in Table 40 on page 344 and Table 41 on page 345.

Discussions in the following section use the terms **volume** and **reel**. The term **volume** refers to all non-unit-record input-output devices. The term **reel** applies only to tape devices. Treatment of direct access devices in the sequential access mode is logically equivalent to the treatment of tape devices.

## Common Processing Facilities

There are several common processing facilities that apply to more than one input-output statement. The common processing facilities provided are:

- Status key
- Invalid key condition
- INTO/FROM identifier phrase
- File position indicator

### Status Key

If the FILE STATUS clause is specified in the FILE-CONTROL entry, a value is placed in the specified status key (the 2-character data item named in the FILE STATUS clause) during execution of any request on that file; the value indicates the status of that request. The value is placed in the status key before execution of any EXCEPTION/ERROR declarative or INVALID KEY/AT END phrase associated with the request.

There are two status key data-names. One is described by data-name-1 in the FILE STATUS clause of the FILE-CONTROL entry. This is a two character data item with the first character known as status key 1 and the second character known as status key 2. The combinations of possible values and their meanings are shown in Table 32 on page 255.

The other status key is described by data-name-8 in the FILE STATUS clause of the FILE-CONTROL entry. Data-name-8 does not apply to QSAM files (MVS and VM only) or to line sequential files (Workstation only). For more information on data-name-8, see "FILE STATUS Clause" on page 119.

*Table 32 (Page 1 of 4). Status Key Values and Meanings*

| High-Order Digit | Meaning | Low-Order Digit | Meaning |
|---|---|---|---|
| 0 | Successful Completion | 0 | No further information |
| | | 2 | This file status value only applies to indexed files with alternate keys that allow duplicates.<br>The input-output statement was successfully executed, but a duplicate key was detected. For a READ statement the key value for the current key of reference was equal to the value of the same key in the next record within the current key of reference. 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.<br>File status 02 does not apply to OS/2 Btrieve** files or STL files. |
| | | 4 | A READ statement was successfully executed, but the length of the record being processed did not conform to the fixed file attributes for that file. |
| | | 5 | An OPEN statement is successfully executed but the referenced optional file is not present at the time the OPEN statement is executed. The file has been created if the open mode is I-O or EXTEND. This does not apply to MVS and VM VSAM sequential files. |
| | | 7 | For a CLOSE statement with the NO REWIND, REEL/UNIT, or FOR REMOVAL phrase or for an OPEN statement with the NO REWIND phrase, the referenced file was on a non-reel/unit medium. |
| 1 | At end condition | 0 | A sequential READ statement was attempted and no next logical record existed in the file because the end of the file had been reached, or the first READ was attempted on an optional input file that was not present. |
| | | 4 | A sequential READ statement was attempted for a relative file and the number of significant digits in the relative record number was larger than the size of the relative key data item described for the file. |

*Table 32 (Page 2 of 4). Status Key Values and Meanings*

| High-Order Digit | Meaning | Low-Order Digit | Meaning |
|---|---|---|---|
| 2 | Invalid key condition | 1 | A sequence error exists for a sequentially accessed indexed file. The prime record key value has been changed by the program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file, or the ascending requirements for successive record key values were violated.<br>Under OS/2 for Btrieve indexed files, file status 21 is not applicable. You can create records sequentially in any key order. |
| | | 2 | An attempt was made to write a record that would create a duplicate key in a relative file; or an attempt was made to write or rewrite a record that would create a duplicate prime record key or a duplicate alternate record key without the DUPLICATES phrase in an indexed file. |
| | | 3 | An attempt was made to randomly access a record that does not exist in the file, or a START or random READ statement was attempted on an optional input file that was not present. |
| | | 4 | An attempt was made to write beyond the externally defined boundaries of a relative or indexed file. Or, a sequential WRITE statement was attempted for a relative file and the number of significant digits in the relative record number was larger than the size of the relative key data item described for the file. |
| 3 | Permanent error condition | 0 | No further information |
| | | 4 | A permanent error exists because of a boundary violation; an attempt was made to write beyond the externally-defined boundaries of a sequential file. |
| | | 5 | An OPEN statement with the INPUT, I-O, or EXTEND phrase was attempted on a non-optional file that was not present. |
| | | 7 | An OPEN statement was attempted on a file that would not support the open mode specified in the OPEN statement. Possible violations are:<br>1. The EXTEND or OUTPUT phrase was specified but the file would not support write operations.<br>2. The I-O phrase was specified but the file would not support the input and output operations permitted.<br>3. The INPUT phrase was specified but the file would not support read operations. |
| | | 8 | An OPEN statement was attempted on a file previously closed with lock. |
| | | 9 | The OPEN statement was unsuccessful because a conflict was detected between the fixed file attributes and the attributes specified for that file in the program. These attributes include the organization of the file (sequential, relative, or indexed), the prime record key, the alternate record keys, the code set, the maximum record size, the record type (fixed or variable), and the blocking factor.<br>Under AIX, OS/2, and Windows, file status 39 is not supported for line sequential files or Btrieve files. |

*Table 32 (Page 3 of 4). Status Key Values and Meanings*

| High-Order Digit | Meaning | Low-Order Digit | Meaning |
|---|---|---|---|
| 4 | Logic error condition | 1 | An OPEN statement was attempted for a file in the open mode. |
| | | 2 | A CLOSE statement was attempted for a file not in the open mode. |
| | | 3 | For a mass storage file in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of a REWRITE statement was not a successfully executed READ statement.<br>For relative and indexed files in the sequential access mode, the last input-output statement executed for the file prior to the execution of a DELETE or REWRITE statement was not a successfully executed READ statement. |
| | | 4 | A boundary violation exists because an attempt was made to rewrite a record to a file and the record was not the same size as the record being replaced, or an attempt was made to write or rewrite a record that was larger than the largest or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name. |
| | | 6 | A sequential READ statement was attempted on a file open in the input or I-O mode and no valid next record had been established because:<br>1. The preceding READ statement was unsuccessful but did not cause an at end condition<br>2. The preceding READ statement caused an at end condition. |
| | | 7 | The execution of a READ statement was attempted on a file not open in the input or I-O mode. |
| | | 8 | The execution of a WRITE statement was attempted on a file not open in the I-O, output, or extend mode. |
| | | 9 | The execution of a DELETE or REWRITE statement was attempted on a file not open in the I-O mode. |

*Table 32 (Page 4 of 4). Status Key Values and Meanings*

| High-Order Digit | Meaning | Low-Order Digit | Meaning |
|---|---|---|---|
| 9 | Implementor-defined condition | 0 | No further information. |
| | | 1 | For VSAM only on MVS and VM: Password failure.<br>Under AIX, OS/2, and Windows: Authorization failure. |
| | | 2 | Logic error. |
| | | 3 | For all files, except QSAM: Resource not available. |
| | | 4 | For VSAM under MVS and VM with CMPR2 compiler-option only: No file position indicator for sequential request.<br>Under AIX, OS/2, and Windows: Concurrent open error. |
| | | 5 | For all files, except QSAM: Invalid or incomplete file information. |
| | | 6 | For VSAM file under MVS and VM: No DD statement was specified.<br>For QSAM file under MVS and VM: No DD statement was specified and CBLQDA(OFF) was specified.<br>Under AIX, OS/2, and Windows: File name error. |
| | | 7 | For VSAM only under MVS and VM: OPEN statement execution successful: File integrity verified.<br>Under AIX, OS/2, and Windows: Errors related to remote file access. |
| | | 8 | Under AIX, OS/2, and Windows: Open failed due to locked file. |
| | | 9 | Under AIX, OS/2, and Windows: Record access failed due to locked record. |

## Invalid Key Condition

The invalid key condition can occur during execution of a START, READ, WRITE, REWRITE, or DELETE statement. (For details of the causes for the condition, see the appropriate statement in “Part 4. Environment Division” on page 83.) When an invalid key condition occurs, the input-output statement that caused the condition is unsuccessful.

When the invalid key condition is recognized, actions are taken in the following order:

1. If the FILE STATUS clause is specified in the FILE-CONTROL entry, a value is placed into the status key to indicate an invalid key condition. (See Table 32 on page 255.)
2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any EXCEPTION/ERROR declarative procedure specified for this file is not executed. Execution then continues according to the rules for each statement specified in the imperative-statement.

3. If the INVALID KEY phrase is not specified in the input-output statement for a file, an EXCEPTION/ERROR procedure **must** be specified, and that procedure is executed. The NOT INVALID KEY phrase, if specified, is ignored.

x Both the INVALID KEY phrase and the EXCEPTION/ERROR procedure can be
x omitted.

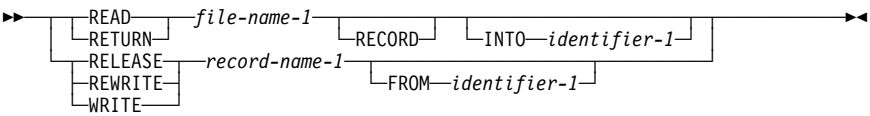
If the invalid key condition does not exist after execution of the input-output operation, the INVALID KEY phrase is ignored, if specified, and the following actions are taken:

1. If an exception condition which is not an invalid key condition exists, control is transferred according to the rules of the USE statement following the execution of any USE AFTER EXCEPTION procedure.
2. If no exception condition exists, control is transferred to the end of the input-output statement or the imperative statement specified in the NOT INVALID KEY phrase, if it is specified.

## INTO/FROM Identifier Phrase

This phrase is valid for READ, RETURN, RELEASE, REWRITE, and WRITE statements. The identifier specified must be the name of an entry in the Working-Storage Section or the Linkage Section, or of a record description for another previously opened file. Record-name/file-name and identifier must not refer to the same storage area.

**Format**




- The INTO phrase can be specified in a READ or RETURN statement.

  The result of the execution of a READ or RETURN statement with the INTO phrase is equivalent to the application of the following rules in the order specified:

  – The execution of the same READ or RETURN statement without the INTO phrase.

  – 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 a group move. The implied MOVE statement does not occur if the execution of the READ or RETURN statement was unsuccessful. Any subscripting or reference-modification associated with identifier-1 is evaluated after the record has been read or returned 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.

- The FROM phrase can be specified in a RELEASE, REWRITE, or WRITE statement.

  The result of the execution of a RELEASE, REWRITE, or WRITE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

  ```
  MOVE identifier-1 TO record-name-1
  ```

  The same RELEASE, REWRITE, or WRITE statement without the FROM phrase.

  After the execution of the RELEASE, REWRITE or WRITE 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 specified by the SAME RECORD AREA clause.

## File Position Indicator

The file position indicator is a conceptual entity used in this document to facilitate exact specification of the next record (or alternatively under AIX, OS/2, and Windows, the previous 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 the OPEN, CLOSE, READ and START statements. The concept of a file position indicator has no meaning for a file opened in the output or extend mode.

Statements, sentences, and paragraphs in the Procedure Division are executed sequentially, except when a procedure **branching** statement such as EXIT, GO TO, PERFORM, GOBACK, or STOP is used.

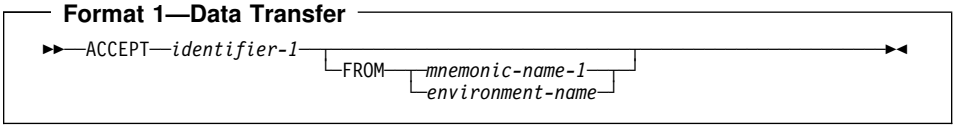
# Procedure Division Statements

## ACCEPT Statement

The ACCEPT statement transfers data into the specified identifier. There is no editing or error checking of the incoming data.

### Data Transfer

**Format 1—Data Transfer**




x

Format 1 transfers data from an input/output device into identifier-1. When the FROM phrase is omitted, the system input device is assumed.

Format 1 is useful for exceptional situations in a program when operator intervention (to supply a given message, code, or exception indicator) is required. The operator must, of course, be supplied with the appropriate messages with which to reply.

WRKSTN Under AIX, OS/2, and Windows, the input file must be a byte stream file (for example, a file consisting of text data with records delimited by a record terminator).
x You can create a byte stream file in your COBOL program using line sequential file I-O
x or with the DISPLAY statement. (Most text editors can be used to create a byte steam file as well.)

The input file **cannot** be a VSAM, Btrieve, or STL file (including sequential, relative, or indexed files).

If the source of the ACCEPT statement is a file and *identifier-1* is filled without using the full record delimited by the record terminator, the remainder of the input record is used in the next ACCEPT statement for the file. The record delimiter characters are removed from the input data before the input records are moved into the ACCEPT receiving area.

If the source of the ACCEPT statement is a terminal, the data entered at the terminal, followed by the enter key, is treated as the input data. If the input data is shorter than *identifier-1*, the area is padded with spaces.

**identifier-1**
Can be any group item, or an elementary alphabetic, alphanumeric, alphanumeric-edited, numeric-edited or external decimal item.

x It can also be a DBCS data item or an external floating-point item.

© Copyright IBM Corp. 1991, 1996

**mnemonic-name**

Must be associated in the SPECIAL-NAMES paragraph with an input/output device: either a system input device or a console. For more information on acceptable values for mnemonic-name, see "SPECIAL-NAMES Paragraph" on page 88.

- System input device

  Record length of 80 characters is assumed even if a logical record length of other than 80 characters is specified.

  The system input device is read until identifier-1 is filled or EOF is encountered. If the length of identifier-1 is not an even multiple of the system input device record length, the final record will be truncated as required. If EOF is encountered after data has been moved, and before identifier-1 has been filled, identifier-1 is padded with blanks. If EOF is encountered before any data has been moved to identifier-1, padding will not take place and identifier-1 contents will remain unchanged. Each input record is concatenated with the previous input record.

  If the input record is of the fixed-length format, the entire input record is used. No editing is performed to remove trailing or leading blanks.

  MVS/VM Under MVS and VM, if the input record is of the variable-length format, the actual record length is used to determine the amount of data received. With variable format records, the Record Definition Word (RDW) is removed from the beginning of the input record. Only the actual input data is transferred to identifier-1.

- Console

  MVS/VM Under MVS and VM:

  1. A system-generated message code is automatically displayed, followed by the literal AWAITING REPLY.

     The maximum length of an input message is 114 characters.

  2. Execution is suspended.

  3. After the message code (the same code as in item 1) is entered from the console and recognized by the system, ACCEPT statement execution is resumed. The message is moved to identifier-1 and left-justified, regardless of its PICTURE clause.

     The ACCEPT statement is terminated after any of the following occurs:

     – If no data is received from the console. For example, if the operator hits the ENTER key

     – The identifier is filled with data

     – Fewer than 114 characters of data are entered.

     If 114 bytes of data are entered and the identifier is still not filled with data, then more requests for data are issued to the console.

If more than 114 characters of data are entered, only the first 114 characters will be recognized by the system.

If the identifier is longer than the incoming message, the rightmost characters are padded with spaces.

If the incoming message is longer than the identifier, the character positions beyond the length of the identifier are truncated. ◁

WRKSTN▷ Under AIX, OS/2, and Windows: ACCEPT or DISPLAY with an environment name is directed to the destination based on the value of the environment variable corresponding to the COBOL environment name (SYSIN or CONSOLE).

If the environment variable corresponding to the COBOL environment name is not set, ACCEPT from SYSIN, SYSIPT, or CONSOLE is from the system logical input device (stdin) and DISPLAY on SYSOUT, SYSLIST, or SYSLST is to the system logical output device (stdout). DISPLAY to SYSPUNCH or SYSPCH is supported only if the environment variable for SYSPUNCH is set to a valid display target. ◁

x **environment-name**

x A valid **environment-name** can be specified. See Table 8 on page 90 for a list of

x valid environment-names.

WRKSTN▷ Under AIX, OS/2, and Windows, ACCEPT (or DISPLAY) with an environment name is directed to the destination based on the assignment of the system target for the environment name via the environment variable assignment.

If the environment variable is not set, the first three are directed to the system logic input device, the system logic output device, and the user terminal, respectively as determined by COBOL for the platform. (For example, CONSOLE would be `stdin` as the default.) DISPLAY to SYSPUNCH/SYSPCH fails unless you set the corresponding environment variable to indicate a valid target. The target file is determined by checking the COBOL environment-name (CONSOLE, SYSIN, SYSIPT, SYSOUT, SYSLIST, SYSLST, SYSPUNCH, and SYSPCH). If an environment variable is defined corresponding to the COBOL environment variable, the value of the environment variable is used as the system file identifier. For more information on environment-variables, see the *IBM COBOL Programming Guide* for your platform. ◁

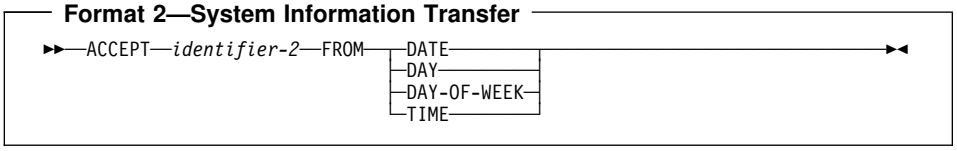
**Note:** If the device is the same as that used for READ statements, results are unpredictable.

## System Information Transfer

System information contained in the specified conceptual data items DATE, DAY, DAY-OF-WEEK, or TIME, can be transferred into the identifier. The transfer must follow the rules for the MOVE statement without the CORRESPONDING phrase. See “MOVE Statement” on page 333.

## ACCEPT Statement

**Format 2—System Information Transfer**




**identifier-1**
Can be a group, elementary alphanumeric, alphanumeric-edited, numeric-edited, external decimal, internal decimal, binary, internal floating-point, or external floating-point item.

Format 2 accesses the current date (in two formats), the day of the week, and the time of day, as carried by the system. This can be useful in identifying when a particular run of an object program was executed. It can also be used to supply the date in headings and footings.

**Note:** the current date and time is also accessible via the date/time intrinsic functions CURRENT-DATE, DAY-OF-INTEGER, DATE-OF-INTEGER, INTEGER-OF-DATE, and INTEGER-OF-DAY (see “Intrinsic Functions” on page 426.) Those functions support 4-digit year values and provide additional information.

## DATE, DAY, DAY-OF-WEEK, and TIME

The conceptual data items DATE, DAY, DAY-OF-WEEK, and TIME implicitly have USAGE DISPLAY. Because DATE, DAY, DAY-OF-WEEK, and TIME are conceptual data items, they cannot be described in the COBOL program.

**DATE**
Has the implicit PICTURE 9(6).

The sequence of data elements (from left to right) is:

```
2 digits for the year
2 digits for the month
2 digits for the day
```

Thus, 27 April 1995 is expressed as: `950427`

**DAY**
Has the implicit PICTURE 9(5).

The sequence of data elements (from left to right) is:

```
2 digits for the year
3 digits for the day
```

Thus, 27 April 1995 is expressed as: `95117`

**DAY-OF-WEEK**

Has the implicit PICTURE 9(1).

The single data element represents the day of the week according to the following values:

```
1 represents Monday                  5 represents Friday
2 represents Tuesday                 6 represents Saturday
3 represents Wednesday               7 represents Sunday
4 represents Thursday
```

Thus, Wednesday is expressed as: `3`

**TIME**

Has the implicit PICTURE 9(8).

The sequence of data elements (from left to right) is:

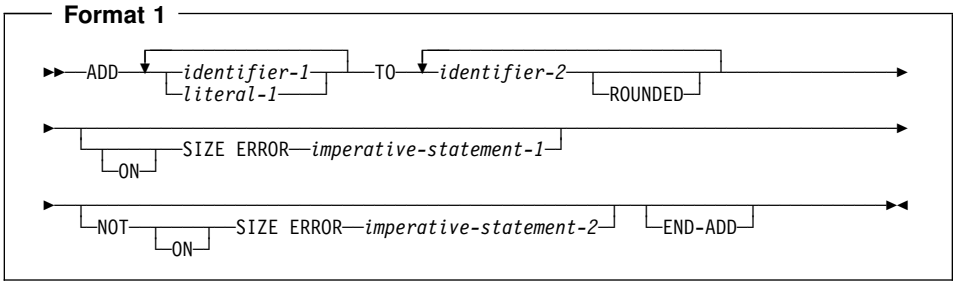
```
2 digits for hour of day
2 digits for minute of hour
2 digits for second of minute
2 digits for hundredths of second
```

Thus, 2:41 PM is expressed as: `14410000`

## ADD Statement

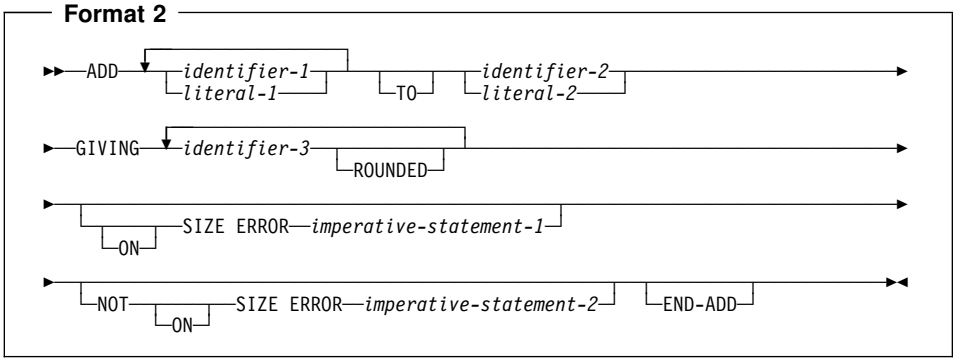
The ADD statement sums two or more numeric operands and stores the result.

**Format 1**



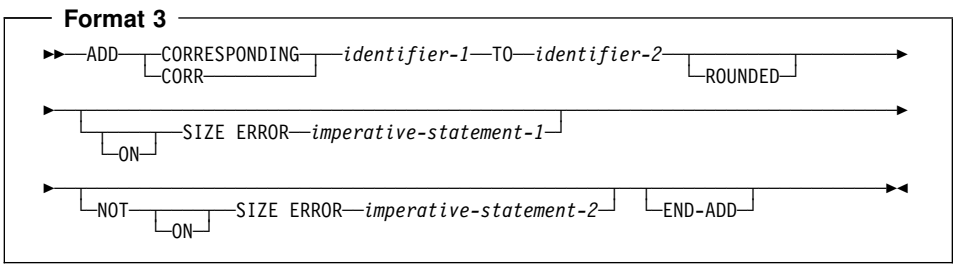

All identifiers or literals preceding the key word TO are added together, and this sum is added to and stored in identifier-2. This process is repeated for each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

**Format 2**




The values of the operands preceding the word GIVING are added together, and the sum is stored as the new value of each data item referenced by identifier-3.

**Format 3**




Elementary data items within identifier-1 are added to and stored in the corresponding elementary items within identifier-2.

For all Formats:

**identifier**
In Format 1, must name an elementary numeric item.

In Format 2, must name an elementary numeric item, except when following the word GIVING. Each identifier following the word GIVING must name an elementary numeric or numeric-edited item.

In Format 3, must name a group item.

**literal**
Must be a numeric literal.

x Floating-point data items and literals can be used anywhere a numeric data item or
x literal can be specified.

The composite of operands must not contain more than 18 digits. The compiler ensures that enough places are carried so that no significant digits are lost during execution.

x The composite of operands can be more than 18 digits. For information on arithmetic
x intermediate results, see the *IBM COBOL Programming Guide* for your platform.

- In Format 1, the composite of operands is determined by using all of the operands in a given statement.
- In Format 2, the composite of operands is determined by using all of the operands in a given statement excluding the data items that follow the word GIVING.
- In Format 3, the composite of operands is determined separately for each pair of corresponding data items.

## ROUNDED Phrase

For Formats 1, 2, and 3, see "ROUNDED Phrase" on page 250.

## SIZE ERROR Phrases

For Formats 1, 2, and 3, see "SIZE ERROR Phrases" on page 251.

## CORRESPONDING Phrase (Format 3)

See "CORRESPONDING Phrase" on page 249.

## END-ADD Phrase

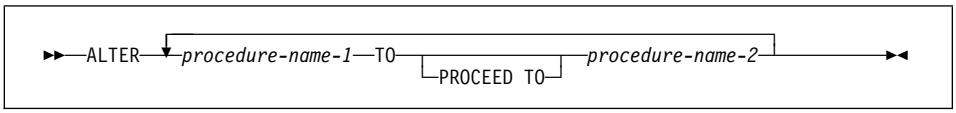
This explicit scope terminator serves to delimit the scope of the ADD statement. END-ADD permits a conditional ADD statement to be nested in another conditional statement. END-ADD can also be used with an imperative ADD statement.

For more information, see "Delimited Scope Statements" on page 247.

## ALTER Statement

The ALTER statement changes the transfer point specified in a GO TO statement.

The ALTER statement encourages the use of unstructured programming practices; the EVALUATE statement provides the same function as the ALTER statement and helps to ensure that your program will be well-structured.




The ALTER statement modifies the GO TO statement in the paragraph named by procedure-name-1. Subsequent executions of the modified GO TO statement(s) transfer control to procedure-name-2.

**procedure-name-1**
Must name a Procedure Division paragraph that contains only one sentence: a GO TO statement without the DEPENDING ON phrase.

**procedure-name-2**
Must name a Procedure Division section or paragraph.

Before the ALTER statement is executed, when control reaches the paragraph specified in procedure-name-1, the GO TO statement transfers control to the paragraph specified in the GO TO statement. After execution of the ALTER statement, however, the next time control reaches the paragraph specified in procedure-name-1, the GO TO statement transfers control to the paragraph specified in procedure-name-2.

The ALTER statement acts as a program switch, allowing, for example, one sequence of execution during initialization and another sequence during the bulk of file processing.

Altered GO TO statements in programs with the INITIAL attribute are returned to their initial states each time the program is entered.

x Do not use the ALTER statement in programs that have the RECURSIVE attribute, in
x methods, or in AIX, OS/2, or Windows programs compiled with the THREAD option.

## Segmentation Considerations

A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority. All other uses of the ALTER statement are valid and are performed, even if the GO TO to which the ALTER refers is in a fixed overlayable segment.

Altered GO TO statements in independent segments are returned to their initial states when control is transferred to the independent segment that contains the ALTERED GO TO from another independent segment with a different priority.

This transfer of control can take place because of:

- The effect of previous statements
- An explicit transfer of control with a PERFORM or GO TO statement
- A sort or merge statement with the INPUT or OUTPUT phrase specified.

## CALL Statement

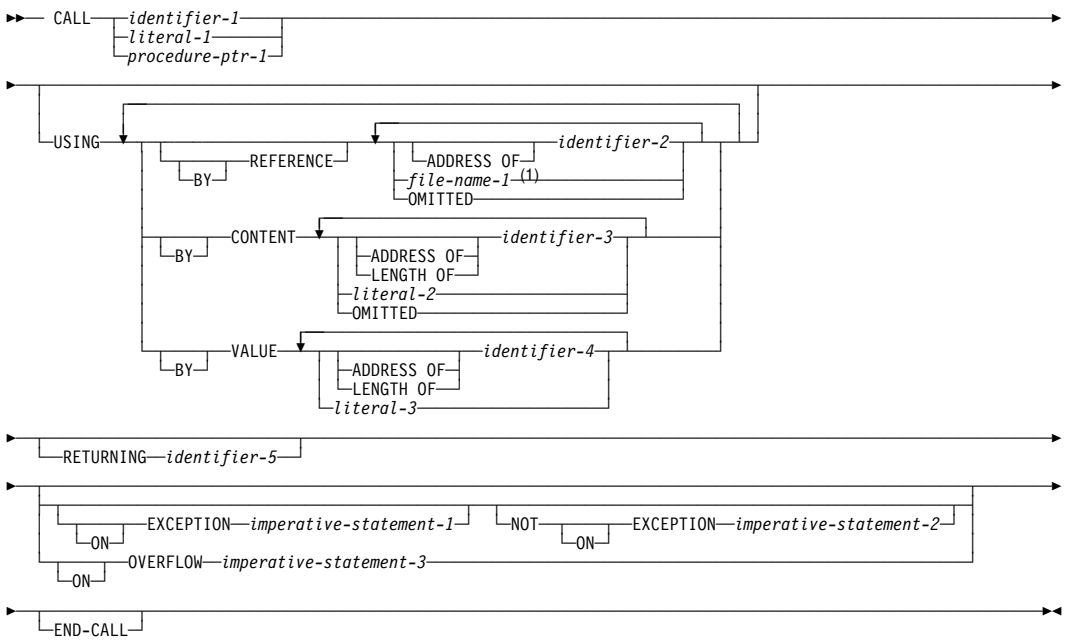
The CALL statement transfers control from one object program to another within the run unit.

The program containing the CALL statement is the calling program; the program identified in the CALL statement is the called subprogram. Called programs can contain CALL statements; however, a called program must not execute a CALL statement that directly or indirectly calls the calling program.

x Programs defined with the RECURSIVE attribute can execute a CALL statement that
x directly or indirectly CALLs itself.

x Do not specify the name of a class or a method in the CALL statement.

**Format**




**Note:**

x [1] *file-name-1* is supported on MVS and VM only.

**identifier-1, literal-1**

Literal-1 must be a nonnumeric literal. Identifier-1 must be an alphanumeric data
x item such that its value can be a program name. The rules of formation for
x program names are dependent on the PGMNAME compiler option. For details,
x see the discussion of program names in "PROGRAM-ID Paragraph" on page 75
x and also the description of the PGMNAME compiler option in the *IBM COBOL Pro-*
x *gramming Guide* for your platform.

x Identifier-1 can be an alphabetic or zoned decimal data item

x **procedure-pointer-1**
x Must be defined with USAGE IS PROCEDURE-POINTER, and must be set to a
x valid program entry point; otherwise, the results of the CALL statement are unde-
x fined.

x After a program has been canceled by COBOL, released by PL/I or C, or deleted
x by assembler, any procedure-pointers that had been set to that program's entry
x point are no longer valid.

When the called subprogram is to be entered at the beginning of the Procedure Division, literal-1 or the contents of identifier-1 must specify the program-name of the called subprogram.

x When the called subprogram is entered through an ENTRY statement, literal-1 or the
x contents of identifier-1 must be the same as the name specified in the called subpro-
x gram's ENTRY statement.

For information on how the compiler resolves CALLs to program names found in multiple programs, see “Conventions for Program-Names” on page 66.

## USING Phrase

The USING phrase specifies arguments that are passed to the target program.

Include the USING phrase in the CALL statement only if there is a USING phrase in the
x Procedure Division header or the ENTRY statement through which the called program is invoked. The number of operands in each USING phrase must be identical.

For more information on the USING phrase see “The Procedure Division Header” on page 217.

The sequence of appearance of the identifiers in the USING phrase of the CALL statement and in the corresponding USING phrase in the called subprogram's Procedure Division header determines the correspondence between the identifiers used by the calling and called programs. This correspondence is positional.

x The sequence of appearance of the identifiers in the USING phrase of the CALL state-
x ment and in the corresponding USING phrase in the called program's ENTRY state-
x ment determines the correspondence between the identifiers used by the calling and
x called programs.

The values of the parameters referenced in the USING phrase of the CALL statement are made available to the called subprogram at the time the CALL statement is executed. The description of the data item in the called program must describe the same number of character positions as the description of the corresponding data item in the calling program.

x The BY CONTENT, BY REFERENCE and BY VALUE phrases apply to parameters that
x follow them until another BY CONTENT, BY REFERENCE, or BY VALUE phrase is encountered. BY REFERENCE is assumed if you do not specify a BY CONTENT, BY
x REFERENCE, or BY VALUE phrase prior to the first parameter.

## BY REFERENCE Phrase

If the BY REFERENCE phrase is either specified or implied for a parameter, the corresponding data item in the calling program occupies the same storage area as the data item in the called program.

**identifier-2**

x Can be a data item of any level in the DATA DIVISION. Identifier-2 cannot be a function identifier.

**Note:** If defined in the Linkage Section or File Section, you must have already provided addressability for identifier-2 prior to invocation of the CALL statement. You can do this by coding either one of the following: `SET ADDRESS OF identifier-2 TO pointer` or `PROCEDURE/ENTRY USING`.

x **file-name-1 (MVS and VM Only)**

x MVS/VM Under MVS and VM, a file-name for a QSAM file. See *IBM COBOL for*
x *MVS & VM Programming Guide* for details on using file-name with the CALL state-
x ment.

x **ADDRESS OF Special Register**

x For information on the ADDRESS OF special register, see "ADDRESS OF" on page 13.

x **OMITTED**

x Indicates that no argument is passed.

## BY CONTENT Phrase

If the BY CONTENT phrase is specified or implied for a parameter, the called program cannot change the value of this parameter as referenced in the CALL statement's USING phrase, though the called program can change the value of the data item referenced by the corresponding data-name in the called program's Procedure Division header. Changes to the parameter in the called program do not affect the corresponding argument in the calling program.

**identifier-3**

x Can be a data item of any level in the DATA DIVISION. Identifier-3 cannot be a function identifier.

**Note:** If defined in the Linkage Section or File Section, you must have already provided addressability for identifier-3 prior to invocation of the CALL statement. You can do this by coding either one of the following: `SET ADDRESS OF identifier-3 TO pointer` or `PROCEDURE/ENTRY USING`.

x **literal-2**

x Can be:

x • A nonnumeric literal
x • A figurative constant (except ALL literal or NULL/NULLS)
x • A DBCS literal

**LENGTH OF Special Register**
For information on the LENGTH OF special register, see “LENGTH OF” on page 15.

**ADDRESS OF Special Register**
For information on the ADDRESS OF special register, see “ADDRESS OF” on page 13.

**OMITTED**
Indicates that no argument is passed.

For nonnumeric literals, the called subprogram should describe the parameter as `PIC X(n) USAGE DISPLAY`, where "n" is the number of characters in the literal.

For DBCS literals, the called subprogram should describe the parameter as `PIC G(n) USAGE DISPLAY-1`, or `PIC N(n)` with implicit or explicit USAGE DISPLAY-1, where "n" is the length of the literal.

## BY VALUE Phrase

The BY VALUE phrase applies to all arguments that follow until overridden by another BY REFERENCE or BY CONTENT phrase.

If the BY VALUE phrase is specified or implied for an argument, the value of the argument is passed, not a reference to the sending data item. The called program can modify the formal parameter corresponding to the BY VALUE argument, but any such changes do not affect the argument since the called program has access to a temporary copy of the sending data item.

While BY VALUE arguments are primarily intended for communication with non-COBOL programs (such as C), they can also be used for COBOL-to-COBOL invocations. In this case, BY VALUE must be specified or implied for both the argument in the CALL USING phrase and the corresponding formal parameter in the Procedure Division USING phrase.

**identifier-4**
Must be an elementary data item in the DATA DIVISION. It must be one of the following:

- Binary (USAGE BINARY, COMP, COMP-4, or COMP-5)
- Floating point (USAGE COMP-1 or COMP-2)
- Pointer (USAGE POINTER)
- Procedure-pointer (USAGE PROCEDURE-POINTER)
- Object reference (USAGE OBJECT REFERENCE)
- Single-byte alphanumeric (such as PIC X or PIC A)

The following can also be passed BY VALUE:

- Reference modified item with length one
- SHIFT-IN and SHIFT-OUT special registers
- LINAGE-COUNTER special register when it is usage binary

**ADDRESS OF Special Register**

An ADDRESS OF special register passed BY VALUE is treated as a pointer. For information on the ADDRESS OF special register, see “ADDRESS OF” on page 13.

**LENGTH OF Special Register**

A LENGTH OF special register passed BY VALUE is treated as a PIC 9(9) binary. For information on the LENGTH OF special register, see “LENGTH OF” on page 15.

**literal-3**

Must be one of the following:

- Numeric literal
- ZERO
- One-character nonnumeric literal
- Symbolic character
- Single byte figurative constant
  - SPACE
  - QUOTE
  - HIGH-VALUE
  - LOW-VALUE

ZERO is treated as a numeric value; a fullword binary zero is passed.

If literal-3 is a fixed point numeric literal, it must have a precision of 9 or less digits. In this case, a fullword binary representation of the literal value is passed.

If literal-3 is a floating point numeric literal, an 8-byte internal floating point (COMP-2) representation of the value is passed.

Literal-3 must not be a DBCS literal.

## RETURNING Phrase

**identifier-5**
The RETURNING data item, which must be defined in the DATA DIVISION. The return value of the CALLed program is implicitly stored into identifier-5.

You can specify the RETURNING phrase for calls to functions written in COBOL, C, or in other programming languages that use C linkage conventions. If you specify the RETURNING phrase on a CALL to a COBOL subprogram:

- The CALLed subprogram must specify the RETURNING phrase on its Procedure Division header.
- Identifier-5 and the corresponding Procedure Division RETURNING identifier in the target program must have the same PICTURE, USAGE, SIGN, SYNCHRONIZE, JUSTIFIED, and BLANK WHEN ZERO clauses (except that currency signs can differ and periods and commas can be interchanged due to the DECIMAL POINT IS COMMA clause).

  When the target returns, its return value is assigned to identifier-5, using either the rules for SET statement, if identifier-6 is USAGE IS INDEX, USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE; otherwise, the rules for the MOVE statement are used.

If an EXCEPTION or OVERFLOW occurs, identifier-5 is not changed. Identifier-5 must not be reference-modified.

The RETURN-CODE special register is not set by execution of CALL statements that include the RETURNING phrase.

## ON EXCEPTION Phrase

An exception condition occurs when the called subprogram cannot be made available. At that time, one of the following two actions will occur:

1. If the ON EXCEPTION phrase is specified, 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 the ON EXCEPTION phrase **is not** specified in the CALL statement, the NOT ON EXCEPTION phrase, if specified, is ignored.

## NOT ON EXCEPTION Phrase

If an exception condition does not occur (that is, the called subprogram can be made available), control is transferred to the called program. After control is returned from the called program, control is transferred to:

- Imperative-statement-2, if the NOT ON EXCEPTION phrase is specified.
- The end of the CALL statement in any other case (if the ON EXCEPTION phrase is specified, it is ignored).

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.

## ON OVERFLOW Phrase

The ON OVERFLOW phrase has the same effect as the ON EXCEPTION phrase.

## END-CALL Phrase

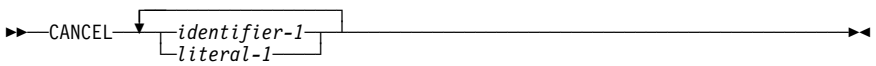
This explicit scope terminator serves to delimit the scope of the CALL statement. END-CALL permits a conditional CALL statement to be nested in another conditional statement. END-CALL can also be used with an imperative CALL statement.

For more information, see “Delimited Scope Statements” on page 247.

## CANCEL Statement

The CANCEL statement ensures that the next time the referenced subprogram is called it will be entered in its initial state.

**Format**




**identifier-1, literal-1**

Literal-1 must be a nonnumeric literal. Identifier-1 must be an alphanumeric data item such that its value can be a program name. The rules of formation for program names are dependent on the PGMNAME compiler option. For details, see the discussion of program names in "PROGRAM-ID Paragraph" on page 75 and also the description of the PGMNAME compiler option in the *IBM COBOL Programming Guide* for your platform.

x Identifier-1 can be alphabetic or zoned decimal data item.

Each literal or contents of the identifier specified in the CANCEL statement must be the same as the literal or contents of the identifier specified in an associated CALL statement.

x The program-name referenced in the CANCEL statement can be affected by the
x PGMNAME compiler option. For details, see the *IBM COBOL Programming Guide*
x for your platform.

x Do not specify the name of a class or a method in the CANCEL statement.

After a CANCEL statement for a called subprogram has been executed, that subprogram no longer has a logical connection to the program. The contents of data items in external data records described by the subprogram are not changed when that subprogram is canceled. If a CALL statement is executed later by any program in the run unit naming the same subprogram, that subprogram will be entered in its initial state.

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 a valid CANCEL were executed for each contained program in the reverse order in which the programs appear in the separately compiled program.

A CANCEL statement closes all open files that are associated with an internal file connector in the program named in the explicit CANCEL statement. Any USE procedures associated with any of these files are not executed.

You can cancel a called subprogram by referencing it as the operand of a CANCEL statement, by terminating the run unit of which the subprogram is a member, or by
x executing an EXIT PROGRAM statement or GOBACK statement in the called subprogram if that subprogram possesses the INITIAL attribute.

## CANCEL Statement

No action is taken when a CANCEL statement is executed, naming a program that either:

1.
   - Under MVS and VM, has not been dynamically called in this run unit by another IBM COBOL for MVS & VM, VS COBOL II, or OS/VS COBOL program.
   - Under AIX, OS/2, and Windows, has not been called in this run unit by another IBM COBOL program.
2. Has been called and subsequently canceled.

Called subprograms can contain CANCEL statements. However, a called program must not execute a CANCEL statement that directly or indirectly cancels the calling program itself, or any other program higher than itself in the calling hierarchy. In such a case, the run unit is terminated.

A program named in a CANCEL statement must not refer to any program that has been
x called and has not yet executed an EXIT PROGRAM or a GOBACK statement.

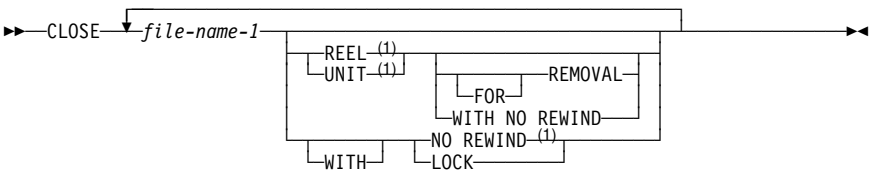
A program can, however, cancel a program that it did not call, providing that, in the calling hierarchy, it is higher than or equal to the program it is canceling.
For example:

```
A calls B and B calls C        (When A receives control,
                                it can cancel C.)

A calls B and A calls C        (When C receives control,
                                it can cancel B.)
```

## CLOSE Statement

The CLOSE statement terminates the processing of volumes and files.

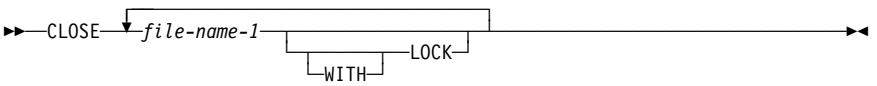
**Format 1—Sequential**



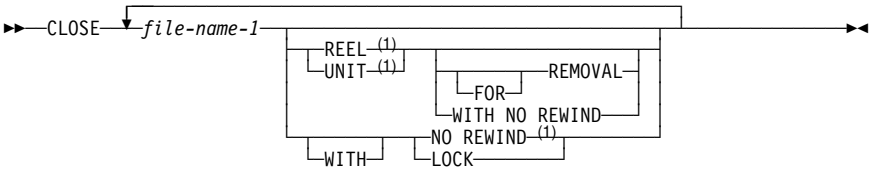

**Note:**

1 Under MVS, the REEL, UNIT, and NO REWIND phrases are not valid for VSAM files. Under AIX, OS/2, and Windows, the UNIT, REEL, and NO REWIND phases are treated as a comment. Although, the file status will be set to 07, indicating a successful completion of a CLOSE for a non-reel/unit medium.

**Format 2—Indexed and Relative Files**




**Format 3—Line Sequential Files (Workstation Only)**




**Note:**

1 Under AIX, OS/2, and Windows, the UNIT, REEL, and NO REWIND phases are treated as a comment. Although, the file status will be set to 07, indicating a successful completion of a CLOSE for a non-reel/unit medium.

**file-name-1**

Designates the file upon which the CLOSE statement is to operate. If more than one file-name is specified, the files need not have the same organization or access. File-name-1 must not be a sort or merge file.

**REEL/UNIT**

Under MVS and VM, you can specify these phrases only for QSAM multivolume or single volume files. The terms REEL and UNIT are interchangeable.

Under AIX, OS/2, and Windows, REEL and UNIT are treated as comments.

## CLOSE Statement

**WITH NO REWIND and FOR REMOVAL**

Under MVS and VM, these phrases apply only to QSAM tape files. If they are specified for storage devices to which they do not apply, they are ignored.

Under AIX, OS/2, and Windows, WITH NO REWIND and FOR REMOVAL are treated as comments.

A CLOSE statement can be executed only for a file in an open mode. After successful execution of a CLOSE statement (without the REEL/UNIT phrase if using format 1):

- The record area associated with the file-name is no longer available. Unsuccessful execution of a CLOSE statement leaves availability of the record data undefined.
- An OPEN statement for the file must be executed before any other input/output statement.
- Under AIX, OS/2, and Windows any record locks and file locks held by the file connector on the closed file are released.

If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the CLOSE statement is executed.

If the file is in an open status and the execution of a CLOSE statement is unsuccessful, the EXCEPTION/ERROR procedure (if specified) for this file is executed.

## Effect of CLOSE Statement on File Types

If the SELECT OPTIONAL clause is specified in the FILE-CONTROL entry for a file, and the file is not present at run time, standard end-of-file processing is not performed. For QSAM files, the file position indicator and current volume pointer are unchanged.

Files are divided into the following types:

**Non-Reel/Unit**

A file whose input or output medium is such that rewinding, reels, and units have no meaning. All VSAM, Btrieve, and STL files are non-reel/unit file types. QSAM files can be non-reel/unit file types.

**Sequential Single Volume**

A sequential file that is contained entirely on one volume. More than one file can be contained on this volume. All VSAM, Btrieve, and STL files are single volume. QSAM files can be single volume.

**Sequential Multivolume**

A sequential file that is contained on more than one volume. QSAM files are the only files that can be multivolume. The concept of volume has no meaning for VSAM, Btrieve, or STL files.

The permissible combinations of CLOSE statement phrases are included in:

- Table 33 on page 281 for sequential files
- Table 34 on page 281 for indexed and relative files
- Table 35 on page 281 for line sequential files

The meaning of each key letter is shown in Table 36 on page 281.

*Table 33. Sequential Files and CLOSE Statement Phrases*

| CLOSE Statement Phrases | Non-Reel/ Unit | Sequential Single-Volume | Sequential Multi-Volume |
|---|---|---|---|
| CLOSE | C | C, G | A, C, G |
| CLOSE REEL/UNIT | F | F, G | F, G |
| CLOSE REEL/UNIT WITH NO REWIND | F | B, F | B, F |
| CLOSE REEL/UNIT FOR REMOVAL | D | D | D |
| CLOSE WITH NO REWIND | C, H | B, C | A, B, C |
| CLOSE WITH LOCK | C, E | C, E, G | A, C, E, G |

*Table 34. Indexed and Relative File Types and CLOSE Statement Phrases*

| CLOSE Statement Phrases | Action |
|---|---|
| CLOSE | C |
| CLOSE WITH LOCK | C,E |

*Table 35. Line Sequential File Types and CLOSE Statement Phrases*

| CLOSE Statement Phrases | Action |
|---|---|
| CLOSE | C |
| CLOSE WITH LOCK | C,E |

*Table 36 (Page 1 of 2). Meanings of Key Letters for Sequential File Types*

| Key | Actions Taken |
|---|---|
| A | **Previous Volumes Unaffected**<br>**Input and Input-Output Files**—Standard volume-switch processing is performed for all previous volumes (except those controlled by a previous CLOSE REEL/UNIT statement). Any subsequent volumes are not processed.<br>**Output Files**—Standard volume-switch processing is performed for all previous volumes (except those controlled by a previous CLOSE REEL/UNIT statement). |
| B | **No Rewinding of Current Reel**—the current volume is left in its current position. |

*Table 36 (Page 2 of 2). Meanings of Key Letters for Sequential File Types*

| Key | Actions Taken |
|---|---|
| C | **Close File**<br><br>**Input and Input-Output Files**—If the file is at its end, and label records are specified, the standard ending label procedure is performed. Standard system closing procedures are then performed.<br><br>If the file is at its end, and label records are not specified, label processing does not take place, but standard system closing procedures are performed.<br><br>If the file is not at its end, standard system closing procedures are performed, but there is no ending label processing.<br><br>**Output Files**—If label records are specified, standard ending label procedures are performed. Standard system closing procedures are then performed.<br><br>If label records are not specified, ending label procedures are not performed, but standard system closing procedures are performed. |
| D | **Volume Removal**—Treated as a comment. |
| E | **File Lock**—The compiler ensures that this file cannot be opened again during this execution of the object program. |
| F | **Close Volume**<br><br>**Input and Input-Output Files**—If the current reel/unit is the last and/or only reel/unit for the file or if the reel is on a non-reel/unit medium, no volume switching is performed. If another reel/unit exists for the file, the following operations are performed: a volume switch, beginning volume label procedure, and the first record on the new volume is made available for reading. If no data records exist for the current volume, another volume switch occurs.<br><br>**Output (Reel/Unit Media) Files**—The following operations are performed: the ending volume label procedure, a volume switch, and the beginning volume label procedure. The next executed WRITE statement places the next logical record on the next direct access volume available. A close statement with the REEL phrase does not close the output file; only an end-of-volume condition occurs.<br><br>**Output (Non-Reel/Unit Media) Files**—Execution of the CLOSE statement is considered successful. The file remains in the open mode and no action takes place except that the value of the I-O status associated with the file is updated. |
| G | **Rewind**—The current volume is positioned at its physical beginning. |
| H | **Optional Phrases Ignored**—The CLOSE statement is executed as if none of the optional phrases were present. |

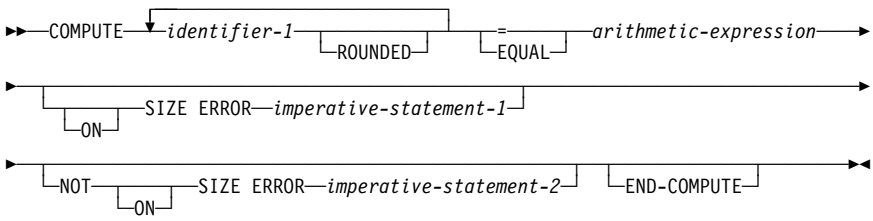
## COMPUTE Statement

The COMPUTE statement assigns the value of an arithmetic expression to one or more data items.

With the COMPUTE statement, arithmetic operations can be combined without the restrictions on receiving data items imposed by the rules for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements.

When arithmetic operations are combined, the COMPUTE statement can be more efficient than the separate arithmetic statements written in a series.

**Format**

x 


**identifier-1**

Must name elementary numeric item(s) or elementary numeric-edited item(s).

x Can name an elementary floating-point data item.

x The word EQUAL can be used in place of =.

**arithmetic-expression**

Can be any arithmetic expression, as defined in “Arithmetic Expressions” on page 223.

When the COMPUTE statement is executed, the value of the arithmetic expression is calculated, and this value is stored as the new value of each data item referenced by identifier-1.

An arithmetic expression consisting of a single identifier, numeric function, or literal allows the user to set the value of the data item(s) referenced by identifier-1 equal to the value of that identifier or literal.

## ROUNDED Phrase

For a discussion of the ROUNDED phrase, see “ROUNDED Phrase” on page 250.

## SIZE ERROR Phrases

For a discussion of the SIZE ERROR phrases, see “SIZE ERROR Phrases” on page 251.

## COMPUTE Statement

## END-COMPUTE Phrase

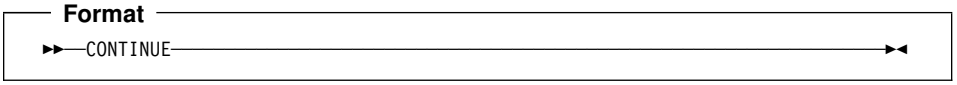
This explicit scope terminator serves to delimit the scope of the COMPUTE statement. END-COMPUTE permits a conditional COMPUTE statement to be nested in another conditional statement. END-COMPUTE can also be used with an imperative COMPUTE statement.

For more information, see “Delimited Scope Statements” on page 247.

## CONTINUE Statement

The CONTINUE statement allows you to specify a no operation statement. CONTINUE indicates that no executable instruction is present.

**Format**

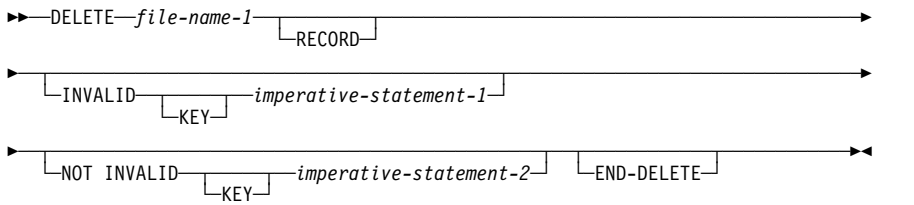


## DELETE Statement

The DELETE statement removes a record from an indexed or relative file. For indexed files, the key can then be reused for record addition. For relative files, the space is then available for a new record with the same RELATIVE KEY value.

When the DELETE statement is executed, the associated file must be open in I-O mode.

**Format**




**file-name-1**
Must be defined in an FD entry in the Data Division and must be the name of an indexed or relative file.

After successful execution of a DELETE statement, the record is removed from the file and can no longer be accessed.

WRKSTN Under AIX, OS/2, and Windows, after the successful execution of a DELETE statement, any record lock held by the file connector on the deleted record is released. However, if any other file connector holds a lock on the record to be deleted, the DELETE statement is unsuccessful.

Execution of the DELETE statement does not affect the contents of the record area associated with file-name-1 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**.

If the FILE STATUS clause is specified in the File-Control entry, the associated status key is updated when the DELETE statement is executed.

The file position indicator is not affected by execution of the DELETE statement.

## Sequential Access Mode

For a file in sequential access mode, the last previous input/output statement must be a successfully executed READ statement. When the DELETE statement is executed, the system removes the record retrieved by that READ statement.

For a file in sequential access mode, the INVALID KEY and NOT INVALID KEY phrases must **not** be specified. However, an EXCEPTION/ERROR procedure can be specified.

## Random or Dynamic Access Mode

In random or dynamic access mode, DELETE statement execution results depend on the file organization: indexed or relative.

When the DELETE statement is executed, the system removes the record identified by the contents of the prime RECORD KEY data item for indexed files, or the RELATIVE KEY data item for relative files. If the file does not contain such a record, an INVALID KEY condition exists. (See "INVALID KEY Condition" under "Common Processing Facilities" on page 254.)

x As an IBM extension, the INVALID KEY phrase does not need to be specified for a
x DELETE statement that references a file in random or dynamic access and for which
x an EXCEPTION/ERROR procedure is not specified.

Transfer of control after the successful execution of a DELETE statement, with the NOT INVALID KEY phrase specified, is to the imperative statement associated with the phrase.

## END-DELETE Phrase

This explicit scope terminator serves to delimit the scope of the DELETE statement. END-DELETE permits a conditional DELETE statement to be nested in another conditional statement. END-DELETE can also be used with an imperative DELETE statement.

For more information, see "Delimited Scope Statements" on page 247.

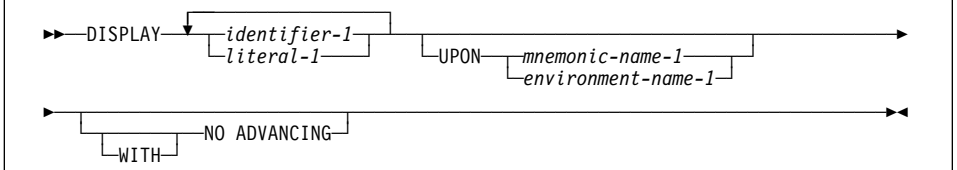
## DISPLAY Statement

The DISPLAY statement transfers the contents of each operand to the output device. The contents are displayed on the output device in the order, left to right, in which the operands are listed.

WRKSTN The target file is determined by checking the COBOL environment-name (CONSOLE, SYSIN, SYSIPT, SYSOUT, SYSLIST, SYSLST, SYSPUNCH, and SYSPCH). If an environment variable is defined corresponding to the COBOL environment-name, the value of the environment-variable is used as the system file identifier. For more information on environment-variables, see the *IBM COBOL Programming Guide* for your platform.

For SYSPUNCH and SYSPCH, the DISPLAY statement will fail unless the corresponding environment variable is set to point to a valid target. ◁□

**Format**




x

**identifier-1**

If it is numeric and is not described as an external decimal, the identifier-1 is converted automatically to external format, as follows:

- Binary or internal decimal items are converted to external decimal. Negative signed values cause a low-order sign overpunch.
- x x Internal floating-point numbers are converted to external floating-point numbers for display, such that:
  - x x – A COMP-1 item will display as if it had an external floating-point PICTURE clause of -.9(8)E-99
  - x x – A COMP-2 item will display as if it had an external floating-point PICTURE clause of -.9(17)E-99

No other identifiers require conversion.

x x Data items defined with USAGE IS POINTER are converted to an external decimal number that would have a PICTURE clause of PIC 9(10).

x x Data items defined with USAGE IS PROCEDURE-POINTER or USAGE IS OBJECT REFERENCE cannot be specified in a DISPLAY statement.

Index names or data items defined with USAGE IS INDEX cannot be specified in a DISPLAY statement.

x DBCS data items, explicitly or implicitly defined as USAGE DISPLAY-1, are transferred
x to the sending field of the output device. Under MVS and VM, for proper results, the
x output device must have the capability to recognize DBCS shift-out and shift-in control
x characters.

x Both DBCS and non-DBCS operands can be specified in a single DISPLAY verb.

**literal-1**

Can be any figurative constant. When a figurative constant is specified, only a single occurrence of that figurative constant is displayed.

Each numeric literal must be an unsigned integer.

x Signed numeric literals and non-integer numeric literals are allowed.

x Floating-point literals are allowed.

x DBCS literals are allowed.

x The ALL figurative constant can be used with DBCS literals in a DISPLAY verb.

**UPON**

x **mnemonic-name** or **environment-name** must be associated in the SPECIAL-NAMES paragraph with an output device. A maximum logical record size is assumed for each device, as follows:

The system logical output device = 120 characters.
The system punch device = 72 characters.
The console = 100 characters.

**Note:** On the system punch device, characters 73 through 80 are used for PROGRAM-ID name.

When the UPON phrase is omitted, the system's logical output device is assumed.

x The list of valid environment-names in a DISPLAY statement is contained in
x Table 8 on page 90.

**WITH NO ADVANCING**

When specified, the positioning of the output device will not be changed in any way following the display of the last operand. If the output 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 output device is not capable of positioning to a specific character position, only the vertical position, if applicable, is affected. This can cause overprinting.

If the WITH NO ADVANCING phrase is not specified, then after the last operand has been transferred to the output device, the positioning of the output device will be reset to the leftmost position of the next line of the device.

The DISPLAY statement transfers the data in the sending field to the output device. The size of the sending field is the total character count of all operands listed. If the output device is capable of receiving data of the same size as the data item being transferred, then the data item is transferred. If the output device is not capable of

receiving data of the same size as the data item being transferred, then one of the following applies:

- If the total character count is less than the device maximum character count, the remaining rightmost characters are padded with spaces.
- If the total character count exceeds the maximum, as many records are written as are needed to display all operands. Any operand being printed or displayed when the end of a record is reached is continued in the next record.

x If a DBCS operand must be split across multiple records, it will be split only on a
x double-byte boundary.

x MVS/VM Under MVS and VM, the shift code compensation is required under this case.
x That is, when a DBCS operand is split across multiple records, the shift-in character
x needs to be inserted at the end of the current record, and the shift-out character needs
x to be inserted at the beginning of the next record. A space is padded after the shift-in
x character, if necessary. These additional inserted shift codes and spaces are included
x in the count while the compiler is calculating the number of records required.

x After the last operand has been transferred to the output device, the device is reset to
x the leftmost position of the next line of the device.

x If a DBCS data item or literal is specified in a DISPLAY verb, the size of the sending
x field is the total character count of all operands listed, with each DBCS character
x counted twice, plus the necessary shift codes for DBCS.

**Notes:**

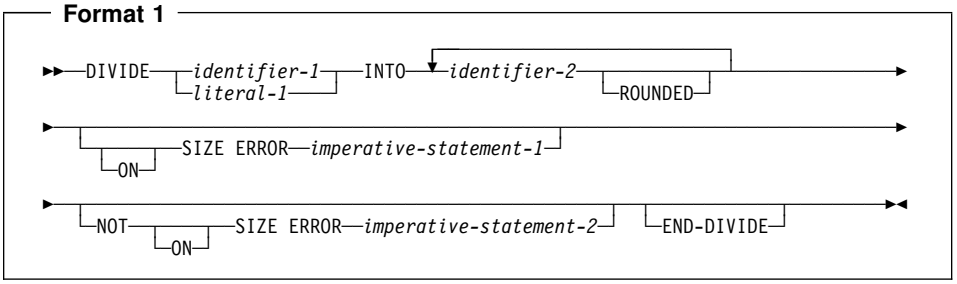
1. The DISPLAY statement causes the printer to space **before** printing.
2. The DISPLAY statement can be used to identify data records that have caused one of the following conditions:
   a. A size error
   b. An invalid key
   c. An overflow condition
   d. A status key returned as a value other than zero

   Such records can be printed, with an identifying message, on some other medium than that used for valid output. Thus, all records for one execution that need special handling are separately printed.

## DIVIDE Statement

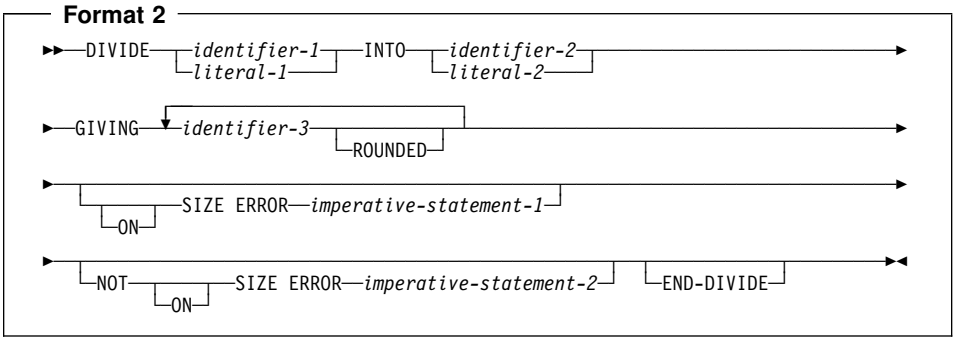
The DIVIDE statement divides one numeric data item into or by other(s) and sets the values of data items equal to the quotient and remainder.

**Format 1**



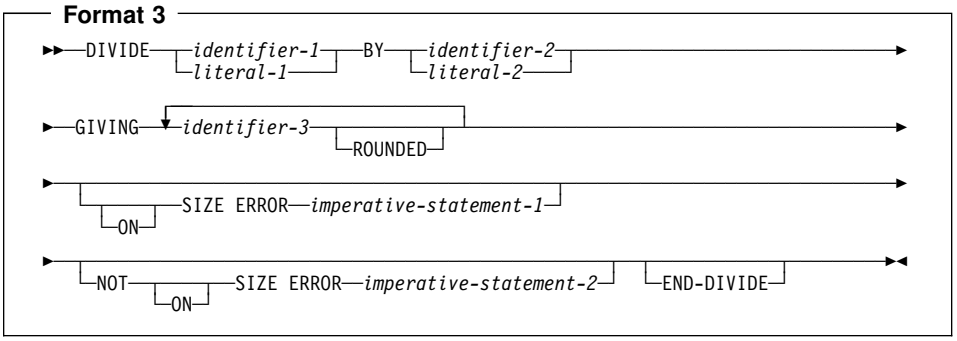

In Format 1, the value of identifier-1 or literal-1 is divided into the value of identifier-2, and the quotient is then stored in identifier-2. For each successive occurrence of identifier-2, the division takes place in the left-to-right order in which identifier-2 is specified.

**Format 2**



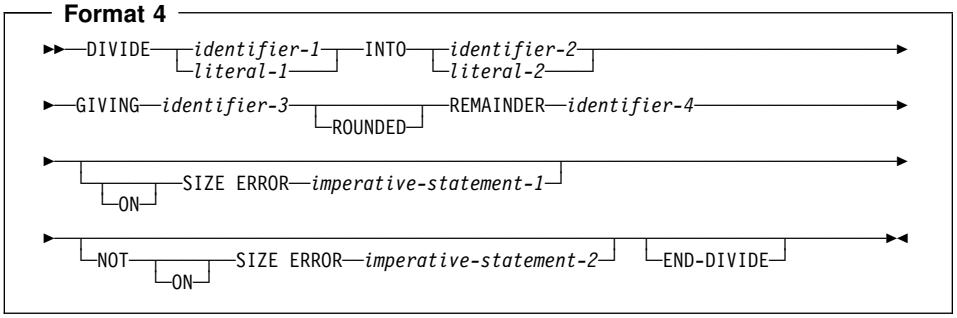

In Format 2, the value of identifier-1 or literal-1 is divided into the value of identifier-2 or literal-2. The value of the quotient is stored in each data item referenced by identifier-3.

**Format 3**



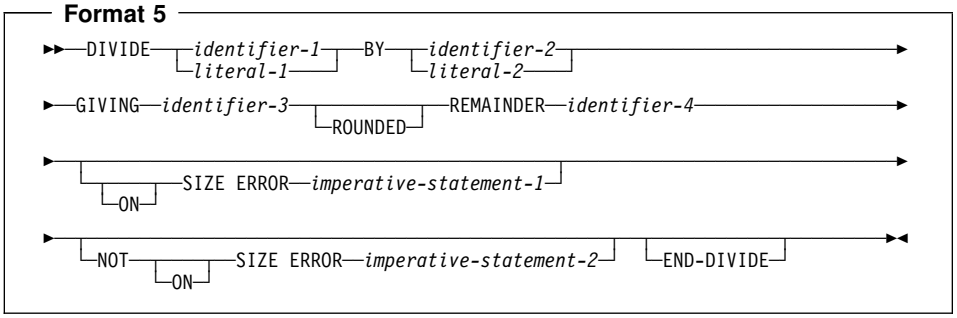
In Format 3, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2. The value of the quotient is stored in each data item referenced by identifier-3.

**Format 4**




In Format 4, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and the value of the remainder is stored in identifier-4.

**Format 5**




In Format 5, the value of identifier-1 or literal-1 is divided by identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and the value of the remainder is stored in identifier-4.

For all Formats:

**identifier-1, identifier-2**
Must name an elementary numeric item.

**identifier-3, identifier-4**
Must name an elementary numeric or numeric-edited item.

**literal-1, literal-2**
Must be a numeric literal.

x In Formats 1, 2, and 3, floating-point data items and literals can be used anywhere that
x a numeric data item or literal can be specified.

x In Formats 4 and 5, floating-point data items or literals cannot be used.

## ROUNDED Phrase

For Formats 1, 2, and 3, see “ROUNDED Phrase” on page 250.

For Formats 4 and 5, the quotient used to calculate the remainder is in an intermediate field. The value of the intermediate field is truncated rather than rounded.

## REMAINDER Phrase

The result of subtracting the product of the quotient and the divisor from the dividend is stored in identifier-4. If identifier-3, the quotient, is a numeric-edited item, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient.

x The REMAINDER phrase is invalid if the receiver or any of the operands is a floating-
x point item.

Any subscripts for identifier-4 in the REMAINDER phrase are evaluated after the result of the divide operation is stored in identifier-3 of the GIVING phrase.

## SIZE ERROR Phrases

For Formats 1, 2, and 3, see “SIZE ERROR Phrases” on page 251.

For Formats 4 and 5, if a size error occurs in the quotient, no remainder calculation is meaningful. Therefore, the contents of the quotient field (identifier-3) and the remainder field (identifier-4) are unchanged.

If size error occurs in the remainder, the contents of the remainder field (identifier-4) are unchanged.

In either of these cases, you must analyze the results to determine which situation has actually occurred.

For information on the NOT ON SIZE ERROR phrase, see page 252.

## END-DIVIDE Phrase

This explicit scope terminator serves to delimit the scope of the DIVIDE statement. END-DIVIDE permits a conditional DIVIDE statement to an imperative statement so that it can be nested in another conditional statement. END-DIVIDE can also be used with an imperative DIVIDE statement.

For more information, see “Delimited Scope Statements” on page 247.

## ENTRY Statement

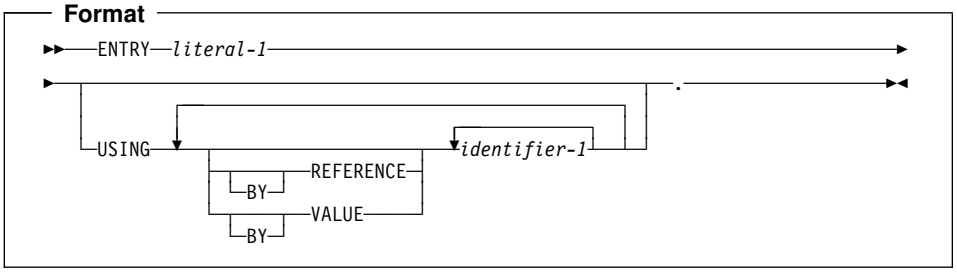
The ENTRY statement establishes an alternate entry point into a COBOL called subprogram.

The ENTRY statement **cannot** be used in:

- Programs that specify a return value using the Procedure Division RETURNING phrase. For details, see the discussion of the RETURNING phrase under "The Procedure Division Header" on page 217.
- Nested program. See "Nested Programs" on page 65 for a description of nested programs.

When a CALL statement naming the alternate entry point is executed in a calling program, control is transferred to the next executable statement following the ENTRY statement.

**Format**




**literal**

Must be nonnumeric and conform to the rules for the formation of a program-name in the outermost program (see "PROGRAM-ID Paragraph" on page 75).

Must not match the program-id or any other ENTRY literal in this program.

Must not be a figurative constant.

Execution of the called program begins at the first executable statement following the ENTRY statement whose literal corresponds to the CALL statement literal or identifier.

The entry point name on the ENTRY statement can be affected by the PGMNAME compiler option. For details, see the *IBM COBOL Programming Guide* for your platform.

## USING Phrase

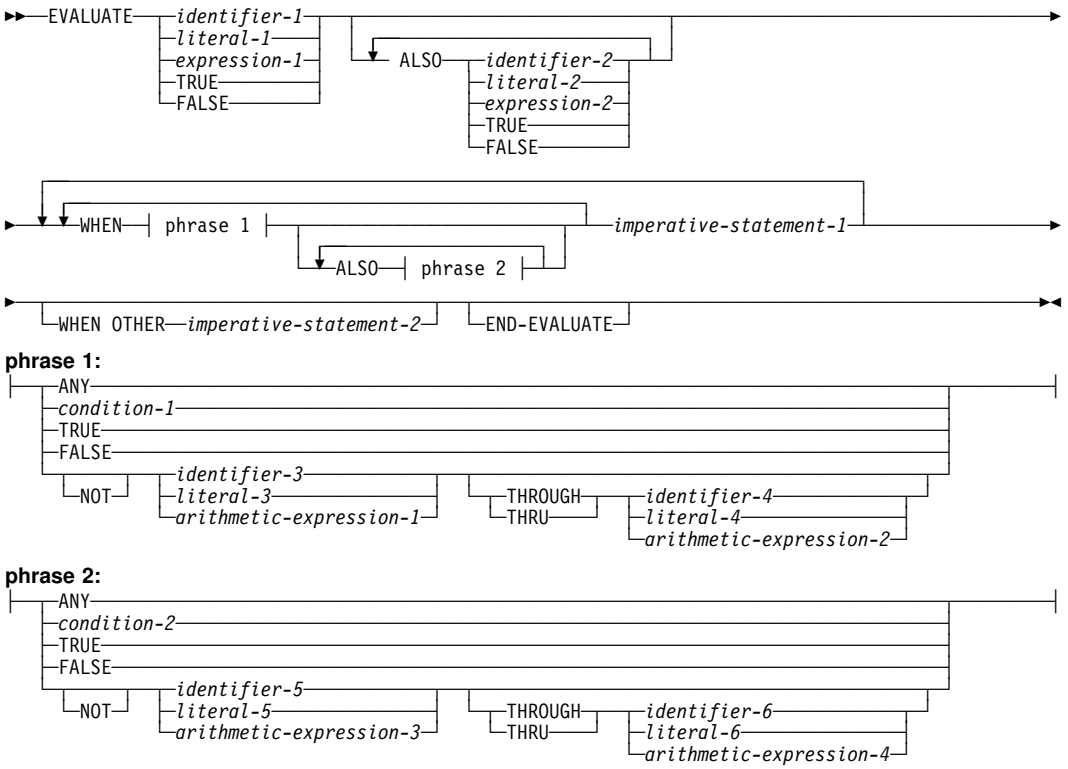
Do not specify the ENTRY statement in a program that contains a Procedure Division ...RETURNING phrase.

For a discussion of the USING phrase, see "The Procedure Division Header" on page 217.

## EVALUATE Statement

The EVALUATE statement provides a shorthand notation for a series of nested IF statements. It can evaluate multiple conditions. That is, the IF statements can be made up of compound conditions. The subsequent action of the object program depends on the results of these evaluations.

**Format**




**Operands before the WHEN phrase**

Are interpreted in one of two ways, depending on how they are specified:

- Individually, they are called selection **subjects**
- Collectively, they are called a **set** of selection subjects.

**Operands in the WHEN phrase**

Are interpreted in one of two ways, depending on how they are specified:

- Individually, they are called selection **objects**
- Collectively, they are called a **set** of selection objects.

**ALSO**

Separates selection subjects within a set of selection subjects; separates selection objects within a set of selection objects.

**THROUGH and THRU**
Are equivalent.

Two operands connected by a THRU phrase must be of the same class. The two operands thus connected constitute a single selection object.

The number of selection objects within each set of selection objects must be equal to the number of selection subjects.

Each selection object within a set of selection objects must correspond to the selection subject having the same ordinal position within the set of selection subjects, according to the following rules:

- Identifiers, literals, or arithmetic expressions appearing within a selection object must be valid operands for comparison to the corresponding operand in the set of selection subjects.
- Condition-1, condition-2, or the word TRUE or FALSE appearing as a selection object must correspond to a conditional expression or the word TRUE or FALSE in the set of selection subjects.
- The word ANY can correspond to a selection subject of any type.
- x Where identifiers are permitted, they can reference DBCS floating-point, USAGE POINTER, USAGE PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE data items.
- x Where nonnumeric literals are permitted, DBCS literals are permitted.
- x Where numeric literals are permitted, floating-point literals are permitted.

## END-EVALUATE Phrase

This explicit scope terminator serves to delimit the scope of the EVALUATE statement. END-EVALUATE permits a conditional EVALUATE statement to be nested in another conditional statement.

For more information, see “Delimited Scope Statements” on page 247.

## Determining Values

The execution of the EVALUATE statement operates as if each selection subject and selection object were evaluated and assigned a numeric or nonnumeric value, a range of numeric or nonnumeric values, or a truth value. These values are determined as follows:

- Any selection subject specified by identifier-1, identifier-2, . . . and any selection object specified by identifier-3 and/or identifier-5 without the NOT or THRU phrase, are assigned the value and class of the data item that they reference.
- Any selection subject specified by literal-1, literal-2, . . . and any selection object specified by literal-3 and/or literal-5 without the NOT or THRU phrase, are assigned the value and class of the specified literal. If literal-3 and/or literal-5 is the figurative constant ZERO, it is assigned the class of the corresponding selection subject.

- Any selection subject in which expression-1, expression-2, . . . is specified as an **arithmetic** expression, and any selection object without the NOT or THRU phrase in which arithmetic-expression-1 and/or arithmetic-expression-3 is specified, are assigned numeric values according to the rules for evaluating an arithmetic expression. (See “Arithmetic Expressions” on page 223.)
- Any selection subject in which expression-1, expression-2, . . . is specified as a **conditional** expression, and any selection object in which condition-1 and/or condition-2 is specified, are assigned a truth value according to the rules for evaluating conditional expressions. (See “Conditional Expressions” on page 225.)
- Any selection subject or any selection object 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.
- Any selection object specified by the word ANY is not further evaluated.
- If the THRU phrase is specified for a selection object without the NOT phrase, the range of values is all values that, when compared to the selection subject, are greater than or equal to the first operand and less than or equal to the second operand, according to the rules for comparison. If the first operand is greater than the second operand, there are no values in the range.
- If the NOT phrase is specified for a selection object, the values assigned to that item are all values not equal to the value, or range of values, that would have been assigned to the item had the NOT phrase been omitted.

## Comparing Selection Subjects and Objects

The execution of the EVALUATE statement then proceeds as if the values assigned to the selection subjects and selection objects were compared to determine whether any WHEN phrase satisfies the set of selection subjects. This comparison proceeds as follows:

1. Each selection object within the set of selection objects for the first WHEN phrase is compared to the selection subject having the same ordinal position within the set of selection subjects. One of the following conditions must be satisfied if the comparison is to be satisfied:
   a. If the items being compared are assigned numeric or nonnumeric values, or a range of numeric or nonnumeric values, the comparison is satisfied if the value, or one value in the range of values, assigned to the selection object is equal to the value assigned to the selection subject, according to the rules for comparison.
   b. If the items being compared are assigned truth values, the comparison is satisfied if the items are assigned identical truth values.
   c. If the selection object being compared is specified by the word ANY, the comparison is always satisfied, regardless of the value of the selection subject.
2. If the above comparison is satisfied for every selection object within the set of selection objects being compared, the WHEN phrase containing that set of selection objects is selected as the one satisfying the set of selection subjects.

3. If the above comparison is not satisfied for every selection object within the set of selection objects being compared, that set of selection objects does not satisfy the set of selection subjects.
4. This procedure is repeated for subsequent sets of selection objects in the order of their appearance in the source program, until either a WHEN phrase satisfying the set of selection subjects is selected or until all sets of selection objects are exhausted.

## Executing the EVALUATE Statement

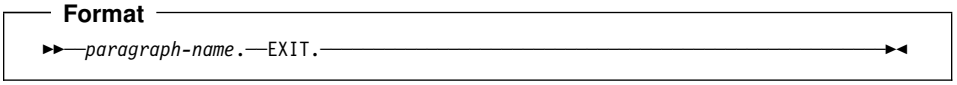
After the comparison operation is completed, execution of the EVALUATE statement proceeds as follows:

- If a WHEN phrase is selected, execution continues with the first imperative-statement-1 following the selected WHEN phrase. Note that multiple WHEN statements are allowed for a single imperative-statement-1.
- If no WHEN phrase is selected and a WHEN OTHER phrase is specified, execution continues with imperative-statement-2.
- If no WHEN phrase is selected and no WHEN OTHER phrase is specified, execution continues with the next executable statement following the scope delimiter.
- The scope of execution of the EVALUATE statement is terminated when execution reaches the end of the scope of the selected WHEN phrase or WHEN OTHER phrase, or when no WHEN phrase is selected and no WHEN OTHER phrase is specified.

## EXIT Statement

The EXIT statement provides a common end point for a series of procedures.

**Format**



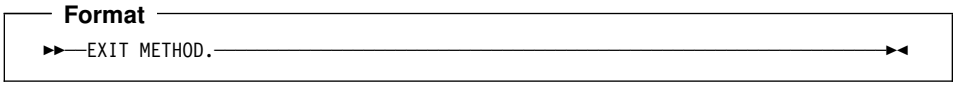

The EXIT statement enables you to assign a procedure-name to a given point in a program.

x As an IBM extension, the EXIT statement does not need to appear in a sentence by
x itself. Any statements following the EXIT statement are executed; the EXIT statement
x is treated as the CONTINUE statement.

## EXIT METHOD Statement

The EXIT METHOD statement specifies the end of an invoked method.

**Format**




You can specify EXIT METHOD only in the Procedure Division of a method. EXIT METHOD causes the executing method to terminate, and control returns to the invoking statement. If the containing method specifies the Procedure Division RETURNING phrase, the value in the data item referred to by the RETURNING phrase becomes the result of the method invocation.

If you need method-specific data to be in the **last-used** state on each invocation, declare it in method Working-Storage. If you need method-specific data to be in the **initial** state on each invocation, declare it in method Local-Storage.

If control reaches an EXIT METHOD statement in a method definition, control returns to the point in the invoking program or method immediately following the INVOKE statement. The state of the invoking program or method is identical to that which existed at the time it executed the INVOKE statement.

The contents of data items and the contents of data files shared between the invoking program or method and the invoked method could have changed. The state of the invoked method is not altered except that the end of the ranges of all PERFORM statement executed by the method are considered to have been reached.

The EXIT METHOD statement does not have to be the last statement in a sequence of imperative statements, but the statements following the EXIT METHOD will not be executed.

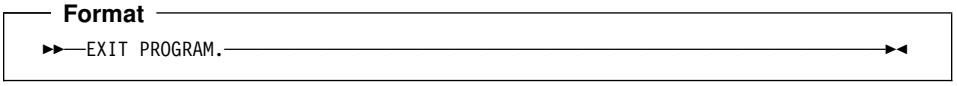
When there is no next executable statement in an invoked method, an implicit EXIT METHOD statement is executed.

## EXIT PROGRAM Statement

The EXIT PROGRAM statement specifies the end of a called program and returns control to the calling program.

You can specify EXIT PROGRAM only in the Procedure Division of a program. It must not be used in a declarative procedure in which the GLOBAL phrase is specified.

**Format**




If control reaches an EXIT PROGRAM statement in a program that does not possess the INITIAL attribute while operating under the control of a CALL statement (that is, the CALL statement is active), control returns to the point in the calling program immediately following the CALL statement. The program state of the calling program is identical to that which existed at the time it executed the CALL statement. The contents of data items and the contents of data files shared between the calling and called program could have been changed. The program state of the called program is not altered except that the ends of the ranges of all PERFORM statements executed by that called program are considered to have been reached.

The execution of an EXIT PROGRAM statement in a called program that possesses the INITIAL attribute is equivalent also to executing a CANCEL statement referencing that program.

If control reaches an EXIT PROGRAM statement, and no CALL statement is active, control passes through the exit point to the next executable statement.

If a subprogram specifies the Procedure Division RETURNING phrase, the value in the data item referred to by the RETURNING phrase becomes the result of the subprogram invocation.

x As an IBM extension, the EXIT PROGRAM statement does not have to be the last
x statement in a sequence of imperative statements, but the statements following the
x EXIT PROGRAM will not be executed if a CALL statement is active.

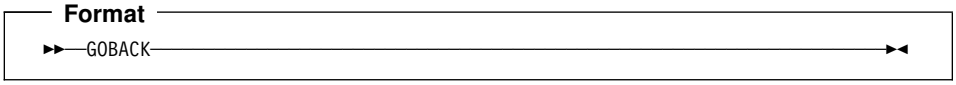
When there is no next executable statement in a called program, an implicit EXIT PROGRAM statement is executed.

## GOBACK Statement

The GOBACK statement functions like the EXIT PROGRAM statement when it is coded as part of a called program (or the EXIT METHOD statement when it is coded as part of an invoked method) and like the STOP RUN statement when coded in a main program.

The GOBACK statement specifies the logical end of a called program or invoked method.

**Format**




A GOBACK statement should appear as the only statement or as the last of a series of imperative statements in a sentence because any statements following the GOBACK are not executed.

If control reaches a GOBACK statement while a CALL statement is active, control returns to the point in the calling program immediately following the CALL statement, as in the EXIT PROGRAM statement.

If control reaches a GOBACK statement while an INVOKE statement is active, control returns to the point in the invoking program or method immediately following the INVOKE statement, as in the EXIT METHOD statement.

In addition, the execution of a GOBACK statement in a called program that possesses the INITIAL attribute is equivalent to executing a CANCEL statement referencing that program.

The table below shows the action taken for the GOBACK statement in both a main program and a subprogram.

| Termination Statement | Main Program | Subprogram |
|---|---|---|
| GOBACK | Return to calling program. (Can be the system and thus causes the job to end.) | Return to calling program. |

## GO TO Statement

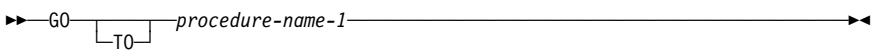
The GO TO statement transfers control from one part of the Procedure Division to another. The types of GO TO statements are:

- Unconditional
- Conditional
- Altered

## Unconditional GO TO

The unconditional GO TO statement transfers control to the first statement in the paragraph or section named in procedure-name, unless the GO TO statement has been modified by an ALTER statement. (See "ALTER Statement" on page 268.)

**Format 1—Unconditional**




**procedure-name-1**
Must name a procedure or a section in the same Procedure Division as the GO TO statement.

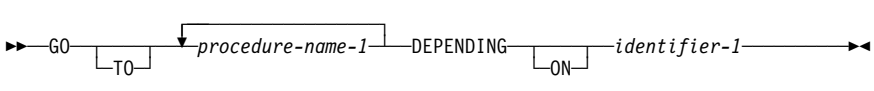
x As an IBM extension, the unconditional GO TO statement does not have to be the last
x statement in a sequence of imperative statements. However, any statements following
x the GO TO are not executed.

When a paragraph is referred to by an ALTER statement, the paragraph must consist of a paragraph-name followed by an unconditional or altered GO TO statement.

## Conditional GO TO

The conditional GO TO statement transfers control to one of a series of procedures, depending on the value of the identifier.

**Format 2—Conditional**




**procedure-name-1**
Must be a procedure or a section in the same Procedure Division as the GO TO statement. The number of procedure-names must not exceed 255.

**identifier-1**
Must be a numeric elementary data item which is an integer.

If 1, control is transferred to the first statement in the procedure named by the first occurrence of procedure-name-1.

If 2, control is transferred to the first statement in the procedure named by the second occurrence of procedure-name-1, and so forth.

If the value of identifier is anything other than a value within the range of 1 through n (where n is the number of procedure-names specified in this GO TO statement), no control transfer occurs. Instead, control passes to the next statement in the normal sequence of execution.

## Altered GO TO

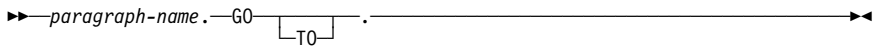
The altered GO TO statement transfers control to the first statement of the paragraph named in the ALTER statement.

You cannot specify the altered GO TO statement in the following:

- A program or method that has the RECURSIVE attribute.
- A program compiled with the THREAD compiler option (Workstation only)

An ALTER statement referring to the paragraph containing an altered GO TO statement must be executed before the GO TO statement is executed. Otherwise, as an IBM extension, the GO TO statement acts like a CONTINUE statement.

**Format 3—Altered**




When an ALTER statement refers to a paragraph, the paragraph can consist only of the paragraph-name followed by an unconditional or altered GO TO statement.

## MORE-Labels GO TO

WRKSTN Under AIX, OS/2, and Windows, GO TO MORE-LABELS is treated as a comment.

The GO TO MORE-LABELS statement can only be specified in a LABEL declarative.

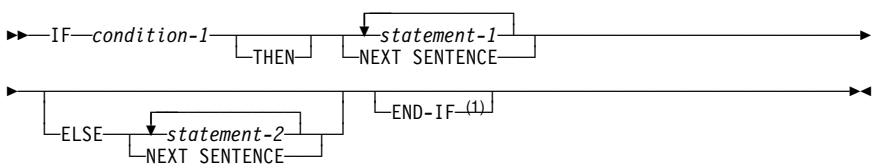
**Format 4—MORE-LABELS**

For more details, see the *IBM COBOL for MVS & VM Programming Guide*.

## IF Statement

The IF statement evaluates a condition and provides for alternative actions in the object program, depending on the evaluation.

**Format**




**Note:**

x [1] END-IF can be specified with NEXT SENTENCE as an IBM extension.

**condition**

Can be any simple or complex condition, as described in “Conditional Expressions” on page 225.

**statement-1, statement-2**

Can be any one of the following:

- An imperative statement
- A conditional statement
- An imperative statement followed by a conditional statement

**NEXT SENTENCE**

If the NEXT SENTENCE phrase is specified, then the END-IF phrase must not be specified.

x END-IF can be specified with NEXT SENTENCE. However, if the NEXT SEN-
x TENCE phrase is executed, control will not pass to the next statement following the
x END-IF but instead will pass to the statement after the closest following period.

## END-IF Phrase

This explicit scope terminator serves to delimit the scope of the IF statement. END-IF permits a conditional IF statement to be nested in another conditional statement. For more information on explicit scope terminators, see “Delimited Scope Statements” on page 247.

The scope of an IF statement can be terminated by any of the following:

- An END-IF phrase at the same level of nesting
- A separator period
- If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting

## Transferring Control

If the condition tested is **true**, one of the following actions takes place:

- If statement-1 is specified, it is executed. If statement-1 contains a procedure branching or conditional statement, control is transferred, according to the rules for that statement. If statement-1 does not contain a procedure-branching statement, the ELSE phrase, if specified, is ignored, and control passes to the next executable statement after the corresponding END-IF or separator period.
- If NEXT SENTENCE is specified, control passes to an implicit CONTINUE statement immediately preceding the next separator period.

If the condition tested is **false**, one of the following actions takes place:

- If ELSE statement-2 is specified, it is executed. If statement-2 contains a procedure-branching or conditional statement, control is transferred, according to the rules for that statement. If statement-2 does not contain a procedure-branching or conditional statement, control is passed to the next executable statement after the corresponding END-IF or separator period.
- If ELSE NEXT SENTENCE is specified, control passes to an implicit CONTINUE STATEMENT immediately preceding the next separator period.
- If neither ELSE statement-2 nor ELSE NEXT STATEMENT is specified, control passes to the next executable statement after the corresponding END-IF or separator period.

**Note:** When the ELSE phrase is omitted, all statements following the condition and preceding the corresponding END-IF or the separator period for the sentence are considered to be part of statement-1.

## Nested IF Statements

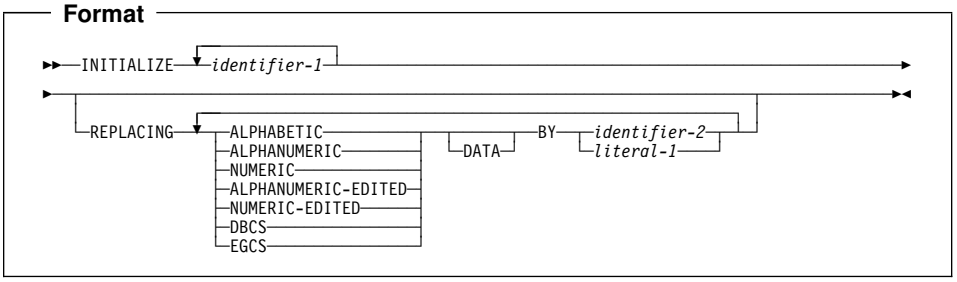
When an IF statement appears as statement-1 or statement-2, or as part of statement-1 or statement-2, it is **nested**.

Nested IF statements (when IF statements contain IF statements) are considered to be matched IF, ELSE, and END-IF combinations proceeding from left to right. Thus, 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.

## INITIALIZE Statement

The INITIALIZE statement sets selected categories of data fields to predetermined values. It is functionally equivalent to one or more MOVE statements.

**Format**




**identifier-1**
Receiving area(s).

**identifier-2, literal-1**
Sending area(s).

A subscripted item can be specified for identifier-1. A complete table can be initialized only by specifying identifier-1 as a group that contains the complete table.

The description of the data item referenced by identifier-1 or any items subordinate to identifier-1 cannot contain the DEPENDING ON phrase of the OCCURS clause. The description of the data item described by identifier-1 can contain the DEPENDING phrase of the OCCURS clause.

**Note:** You cannot use the INITIALIZE statement to initialize a variably located item or group that follows a DEPENDING ON phrase of the OCCURS clause within the same 01 level.

A floating-point data item or literal can be used anywhere a numeric identifier or literal is specified.

A DBCS data item or literal can be used anywhere an identifier or literal is specified.

The data description entry for identifier-1 must not contain a RENAMES clause. An index data item cannot be an operand of INITIALIZE.

Special registers can be specified for identifier-1 and identifier-2 only if they are valid receiving fields or sending fields, respectively, for the implied MOVE statement(s).

## INITIALIZE Statement

## REPLACING Phrase

When the REPLACING phrase is used:

- The category of identifier-2 or literal-1 must be compatible with the category indicated in the corresponding REPLACING phrase, according to the rules for MOVE.
x A floating-point data item or floating-point literal will be treated as if it is in the
x NUMERIC category.
- The same category cannot be repeated in a REPLACING phrase.
- The key word following the word REPLACING corresponds to a category of data shown in "Classes and Categories of Data" on page 136.

When the REPLACING phrase is not used:

- SPACE is the implied sending field for alphabetic, alphanumeric, alphanumeric-
x edited, and DBCS items.
- ZERO is the implied sending field for numeric and numeric-edited items.

x **DBCS**
x **EGCS**
x Refers to the characters allowed for DBCS literals.

## INITIALIZE Statement Rules

1. Whether identifier-1 references an elementary or group item, all operations are performed as if a series of MOVE statements had been written, each of which had an elementary item as a receiving field.

   If the REPLACING phrase is specified:

   - If identifier-1 references a group item, any elementary item within the data item referenced by identifier-1 is initialized only if it belongs to the category specified in the REPLACING phrase.
   - If identifier-1 references an elementary item, that item is initialized only if it belongs to the category specified in the REPLACING phrase.

   This initialization takes place as if the data item referenced by identifier-2 or literal-1 acts as the sending operand in an implicit MOVE statement to the identified item.

   All such elementary receiving fields, including all occurrences of table items within the group, are affected, with the following exceptions:

   - Index data items
   x • Object references
   x • Data items defined with USAGE IS POINTER or USAGE IS
   x PROCEDURE-POINTER
   - Elementary FILLER data items

- Items that are subordinate to identifier-1 and contain a REDEFINES clause, or any items subordinate to such an item. (However, identifier-1 can contain a REDEFINES clause or be subordinate to a redefining item.)

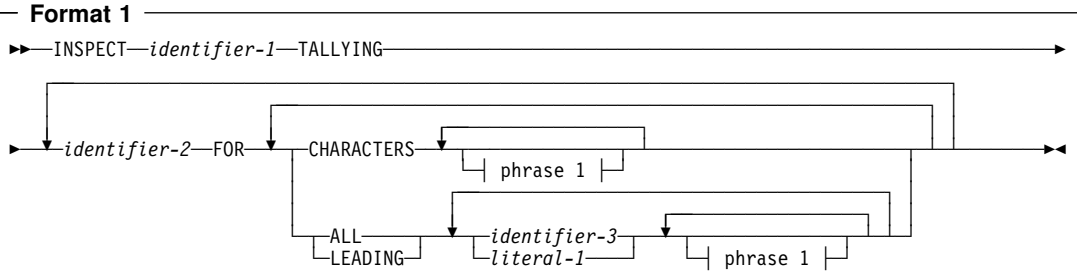
2. The areas referenced by identifier-1 are initialized in the order (left to right) of the appearance of identifier-1 in the statement. Within a group receiving field, affected elementary items are initialized in the order of their definition within the group.
3. If identifier-1 occupies the same storage area as identifier-2, the result of the execution of this statement is undefined, even if these operands are defined by the same data description entry.

## INSPECT Statement

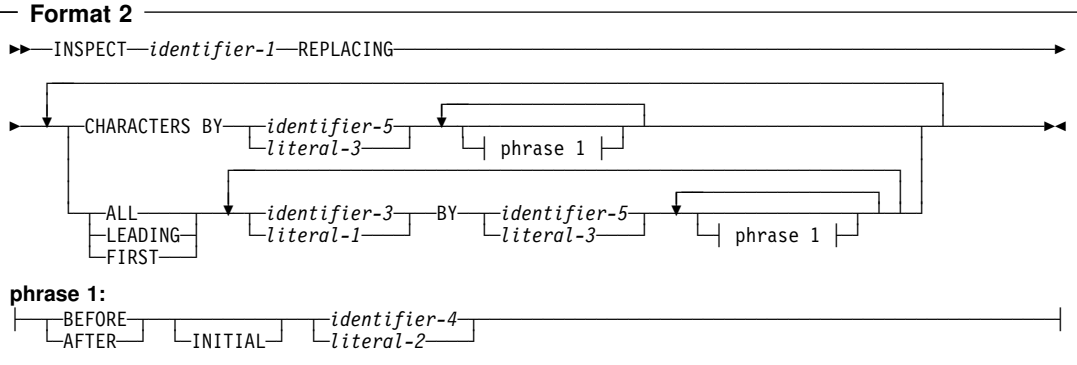
The INSPECT statement specifies that characters, or groups of characters, in a data item are to be counted (tallied) or replaced or both.

- It counts the occurrence of a specific character (alphabetic, numeric, or special character) in a data item. (Formats 1 and 3)
- It fills all or portions of a data item with specified characters, such as spaces or zeros. (Formats 2 and 3)
- It converts all occurrences of specific characters in a data item to user-supplied replacement characters. (Format 4)

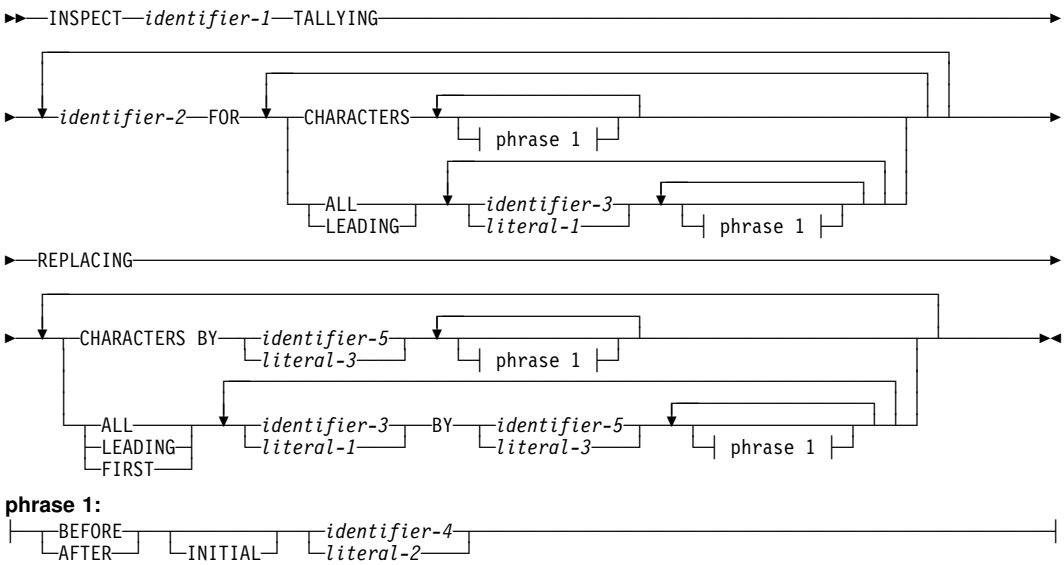
**Format 1**



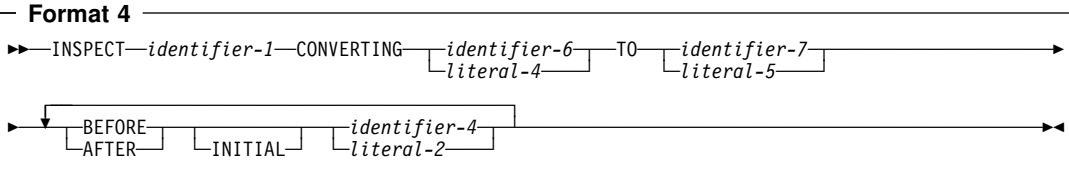

**Format 2**



**Format 3**




**Format 4**




**identifier-1**
Is the **inspected item** and can be either of the following:

- An alphanumeric data item
- A numeric data item with USAGE DISPLAY
- An external floating point item

## Effect of DBCS

All identifiers and literals (except identifier-2) must be DBCS items, either DBCS literals or DBCS data items, if any are DBCS items. Identifier-2 cannot be a DBCS item. DBCS characters, not bytes of data, are tallied in identifier-2.

## TALLYING Phrase (Formats 1 and 3)

This phrase counts the occurrence of a specific character (alphabetic, numeric, or special character) in a data item.

**identifier-2**
Is the **count field**, and must be an elementary integer item defined without the symbol P in its PICTURE character-string.

Identifier-2 cannot be:

- A DBCS item
- An external floating point item

You must initialize identifier-2 before execution of the INSPECT statement begins.

**identifier-3 or literal-1**
Is the **tallying field** (the item whose occurrences will be tallied).

Identifier-3 can be any of the following:

- Elementary alphanumeric data item
- Numeric data item with USAGE DISPLAY
- External floating point item

Literal-1 must be nonnumeric, and can be any figurative constant that does not begin with the word ALL. (If literal-1 is a figurative constant, it is considered to be a 1-character nonnumeric literal.)

**CHARACTERS**
When CHARACTERS is specified and neither the BEFORE nor AFTER phrase is specified, the count field (identifier-2) is increased by 1 for each character (including the space character) in the inspected item (identifier-1). Thus, execution of the INSPECT TALLYING statement increases the value in the count field by the number of characters in the inspected item.

**ALL**
When ALL is specified and neither the BEFORE nor AFTER phrase is specified, the count field (identifier-2) is increased by 1 for each non-overlapping occurrence of the tallying comparand in the inspected item (identifier-1), beginning at the leftmost character position and continuing to the rightmost.

**LEADING**
When LEADING is specified and neither the BEFORE nor AFTER phrase is specified, the count field (identifier-2) is increased by 1 for each contiguous non-overlapping occurrence of the tallying comparand in the inspected item (identifier-1), provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which the tallying comparand is eligible to participate.

**FIRST (Format 3 Only)**
When FIRST is specified and neither the BEFORE nor AFTER phrase is specified, the substitution field replaces the leftmost occurrence of the subject field in the inspected item (identifier-1).

## REPLACING Phrase (Formats 2 and 3)

This phrase fills all or portions of a data item with specified characters, such as spaces or zeros.

**identifier-3 or literal-1**

Is the **subject field** (the item whose occurrences are replaced).

Identifier-3 can be:

- An elementary alphanumeric data item
- A numeric data item with USAGE DISPLAY
- x An external floating point item

Literal-1 must be nonnumeric, and can be any figurative constant that does not begin with the word ALL. If literal-1 is a figurative constant, it is considered to be a 1-character nonnumeric literal.

**identifier-5 or literal-3**

Is the **substitution field** (the item that replaces the subject field).

Identifier-5 can be:

- An elementary alphanumeric data item
- A numeric data item with USAGE DISPLAY
- x An external floating point item

Literal-3 must be nonnumeric, and can be any figurative constant that does not begin with the word ALL.

If literal-3 is a figurative constant, it is considered to be the same length as the subject field.

The subject field and the substitution field must be the same length.

**CHARACTERS BY**

When the CHARACTERS BY phrase is used, the substitution field must be 1 character in length.

When CHARACTERS BY is specified and neither the BEFORE nor AFTER phrase is specified, the substitution field replaces each character in the inspected item (identifier-1), beginning at the leftmost character and continuing to the rightmost.

**ALL**

When ALL is specified and neither the BEFORE nor AFTER phrase is specified, the substitution field replaces each non-overlapping occurrence of the subject field in the inspected item (identifier-1), beginning at the leftmost character position and continuing to the rightmost.

**LEADING**

When LEADING is specified and neither the BEFORE nor AFTER phrase is specified, the substitution field replaces each contiguous non-overlapping occurrence of the subject field in the inspected item (identifier-1), provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which this substitution field is eligible to participate.

**FIRST**
When FIRST is specified and neither the BEFORE nor AFTER phrase is specified, the substitution field replaces the leftmost occurrence of the subject field in the inspected item (identifier-1).

When both the TALLYING and REPLACING phrases are specified (Format 3), the INSPECT statement is executed as if an INSPECT TALLYING statement (Format 1) were specified, immediately followed by an INSPECT REPLACING statement (Format 2).

### Replacement Rules

The following replacement rules apply:

- When the subject field is a figurative constant, the single-character substitution field (which must be 1 character in length) replaces each character in the inspected item equivalent to the figurative constant.
- When the substitution field is a figurative constant, the substitution field replaces each non-overlapping occurrence of the subject field in the inspected item.
- When the subject and substitution fields are character-strings, the character-string specified in the substitution field replaces each non-overlapping occurrence of the subject field in the inspected item.
- After replacement has occurred in a given character position in the inspected item, no further replacement for that character position is made in this execution of the INSPECT statement.

## BEFORE and AFTER Phrases (All Formats)

This phrase narrows the set of items being tallied or replaced.

No more than one BEFORE phrase and one AFTER phrase can be specified for any one ALL, LEADING, CHARACTERS, FIRST or CONVERTING phrase.

**identifier-4 or literal-2**
Is the **delimiter**.

Identifier-4 can be:

- An elementary alphanumeric data item
- A numeric data item with USAGE DISPLAY
- An external floating point item

Literal-2 must be nonnumeric, and can be any figurative constant that does not begin with the word ALL. If literal-2 is a figurative constant, it is considered to be 1 character in length.

Delimiters are not counted or replaced. However, the counting and/or replacing of the inspected item is bounded by the presence of the identifiers and literals.

**INITIAL**
The first occurrence of a specified item.

The BEFORE and AFTER phrases change how counting and replacing are done:

- When BEFORE is specified, counting and/or replacing of the inspected item (identifier-1) begins at the leftmost character and continues until the first occurrence of the delimiter is encountered. If no delimiter is present in the inspected item, counting and/or replacing continues toward the rightmost character.
- When AFTER is specified, counting and/or replacing of the inspected item (identifier-1) begins with the first character to the right of the delimiter and continues toward the rightmost character in the inspected item. If no delimiter is present in the inspected item, no counting or replacement takes place.

## CONVERTING Phrase (Format 4)

This phrase converts all occurrences of specific characters in a data item to user-supplied replacement characters. It can express a string of replacement values.

**identifier-6 or literal-4**
Is the **sending location**.

Identifier-6 can be:

- An elementary alphanumeric data item
- A numeric data item with USAGE DISPLAY
- x An external floating point item

Literal-4 must be nonnumeric, and can be any figurative constant that does not begin with the word ALL. If literal-4 is a figurative constant, it refers to an implicit 1 character data item.

**identifier-7 or literal-5**
Is the **receiving location**.

The receiving location (identifier-7 or literal-5) must be the same size as the sending location (identifier-6 or literal-4).

Identifier-7 can be:

- An elementary alphanumeric data item
- A numeric data item with USAGE DISPLAY
- x An external floating point item.

Literal-5 must be nonnumeric and can be any figurative constant that does not begin with the word ALL. When a figurative constant is used, the size should be equal to the size of literal-4 or identifier-6.

The same character must not appear more than once in either literal-4 or identifier-6.

A Format 4 INSPECT statement is interpreted and executed as if a Format 2 INSPECT statement had been written with a series of ALL phrases (one for each character of literal-4), specifying the same identifier-1. The effect is as if each single character of literal-4 were referenced as literal-1, and the corresponding single character of literal-5 referenced as literal-3. Correspondence between the characters of literal-4 and the characters of literal-5 is by ordinal position within the data item.

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.

## Data Types for Identifiers and Literals

*Table 37. Treatment of the Content of Data Items*

| When referenced by any identifier except identifier-2, the content of each . . . | Is treated . . . |
|---|---|
| alphanumeric or alphabetic item | as a character-string |
| alphanumeric-edited, numeric-edited, or unsigned numeric (external decimal) item | as if redefined as alphanumeric, with the INSPECT statement referring to the alphanumeric item |
| signed numeric (external decimal) item | as if moved to an unsigned external decimal item of the same length and then redefined as alphanumeric, with the INSPECT statement referring to the alphanumeric item.<br>If the sign is a separate character, the byte containing the sign is not examined and, therefore, not replaced. |
| external floating point item | as if redefined as alphanumeric, with the INSPECT statement referring to the alphanumeric item |

## Data Flow

Except when the BEFORE or AFTER phrase is specified, inspection begins at the leftmost character position of the inspected item (identifier-1) and proceeds character-by-character to the rightmost position.

The comparands of the following phrases are compared in the left-to-right order in which they are specified in the INSPECT statement:

- TALLYING (literal-1 or identifier-3, ... )
- REPLACING (literal-3 or identifier-5, ... )

If any identifier is subscripted, reference modified, or is a function-identifier, the subscript, reference-modifier, or function is evaluated only once as the first operation in the execution of the INSPECT statement.

For examples of TALLYING and REPLACING, see the *IBM COBOL Programming Guide* for your platform.

## Comparison Cycle

The comparison cycle consists of the following actions:

1. The first comparand is compared with an equal number of leftmost contiguous characters in the inspected item. The comparand matches the inspected characters only if both are equal, character-for-character.

   If the CHARACTERS phrase is specified, an implied 1-character comparand is used. The implied character is always considered to match the inspected character in the inspected item.

2. If no match occurs for the first comparand and there are more comparands, the comparison is repeated for each successive comparand until either a match is found or all comparands have been acted upon.

3. Depending on whether a match is found, these actions are taken:

   - If a match is found, tallying or replacing takes place, as described in the TALLYING and REPLACING phrase descriptions.

     If there are more characters in the inspected item, the first character following the rightmost matching character is now considered to be in the leftmost character position. The process described in actions 1 and 2 is then repeated.

   - If no match is found and there are more characters in the inspected item, the first character following the leftmost inspected character is now considered to be in the leftmost character position. The process described in actions 1 and 2 is then repeated.

4. Actions 1 through 3 are repeated until the rightmost character in the inspected item either has been matched or has been considered as being in the leftmost character position.

When the BEFORE or AFTER phrase is specified, the comparison cycle is modified, as described in “BEFORE and AFTER Phrases (All Formats)” on page 314.

## Example of the INSPECT Statement

Figure 10 is an example of INSPECT statement results.

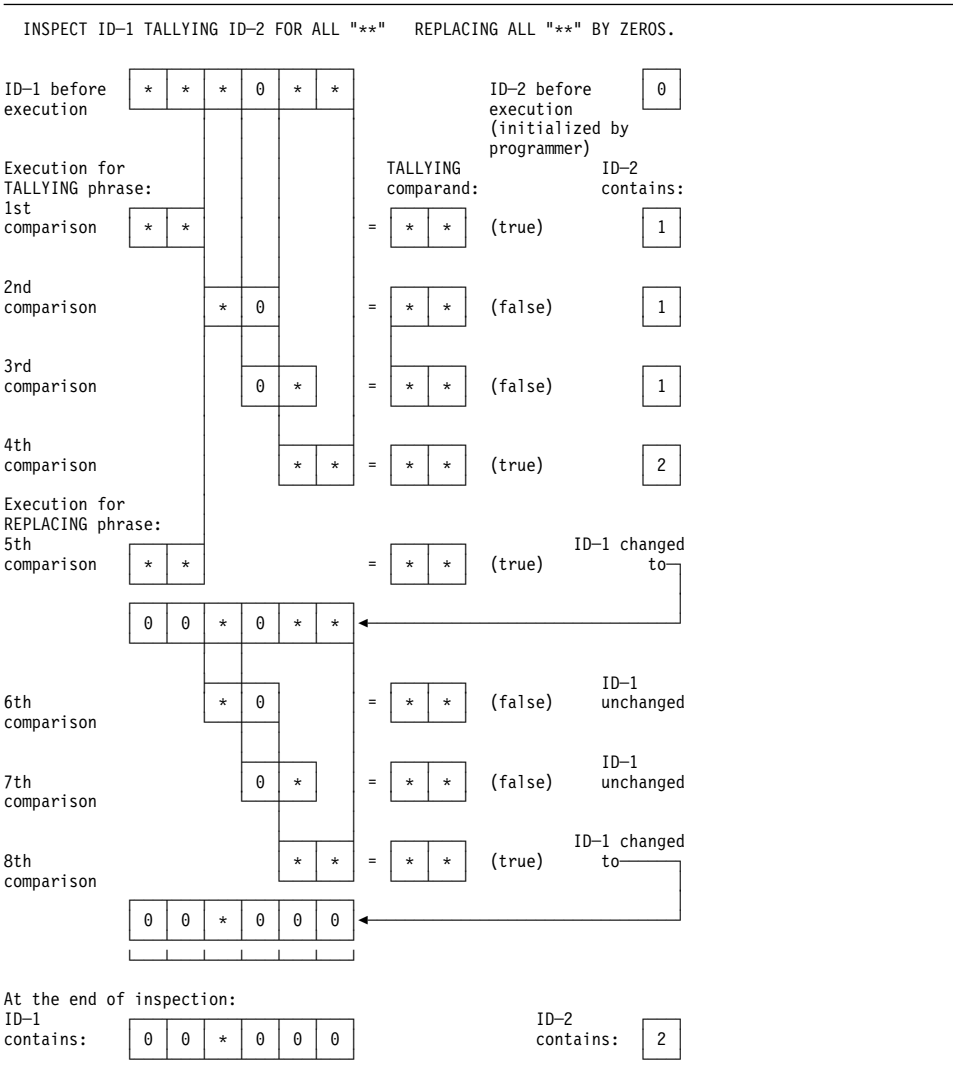


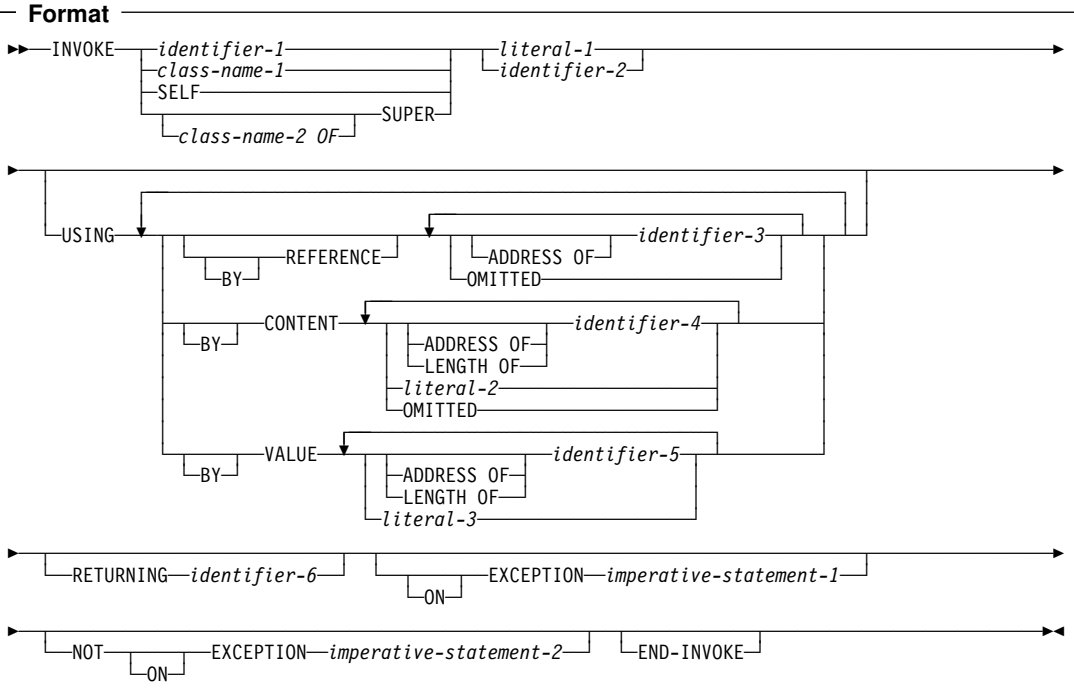

*Figure 10. Example of INSPECT Statement Execution Results*

## INVOKE Statement

The INVOKE statement invokes a method defined in a class or a metaclass. Methods can contain INVOKE statements, and a method can execute an INVOKE statement that directly or indirectly invokes itself. Therefore, all methods are implicitly recursive (unlike COBOL programs, which support recursion only if the RECURSIVE attribute is specified in the PROGRAM-ID paragraph.)

The process for resolving the method name specified on the INVOKE statement to a method implementation is as specified by the IBM SOM. The method resolution is not case sensitive.

**Format**




**identifier-1**

Must be defined as USAGE OBJECT REFERENCE. The contents of identifier-1 specify the object on which a method is invoked.

The results of the INVOKE statement are undefined if:

- identifier-1 does not contain a valid reference to an object or
- identifier-1 contains NULL

**class-name-1**

If class-name-1 is specified, the method is invoked on the class object of class-name-1.

You must specify class-name-1 in the REPOSITORY paragraph of the Configuration Section of the class or program that contains the INVOKE statement.

**SELF**
An implicit reference to the object upon which the currently executing method was invoked. When SELF is specified, the INVOKE statement must appear within the Procedure Division of a method.

**SUPER**
A reference to the object that was used in the invocation of the currently executing method. The resolution of the method to be invoked will ignore any methods declared in the class definition of the currently executing method and methods defined in any class derived from that class, thus the method invoked will be one that is inherited from an ancestor class. To invoke a method that is inherited from a specific class, you can qualify SUPER with a class name (class-name-2).

**class-name-2**
The class to which the inherited method belongs.

You must specify class-name-2 if the reference to SUPER is in a method definition of a class that uses multiple inheritance. Specify class-name-2 in the REPOSITORY paragraph of the Configuration Section of the class or program that contains the INVOKE statement. Class-name-2 must be a direct or indirect parent class.

**literal-1**
The name of the method to be invoked. The referenced object must support the method identified by literal-1.

Literal-1 must be a nonnumeric literal.

**identifier-2**
A nonnumeric data item whose value is a method name. The referenced object must support the method identified by identifier-2.

If identifier-2 is specified, identifier-1 must be defined as USAGE OBJECT REFERENCE without any optional phrases.

## USING Phrase

The USING phrase specifies arguments that are passed to the target method. For details on the USING phrase, see “The Procedure Division Header” on page 217.

For conformance requirements for the USING phrase, see “Conformance Requirements for USING Phrase” on page 323.

## BY REFERENCE Phrase

If the BY REFERENCE phrase is either specified or implied for a parameter, the corresponding data item in the invoking method occupies the same storage area as the data item in the invoked method.

**identifier-3**
Can be a data item of any level in the DATA DIVISION. Identifier-3 cannot be a function-identifier.

If defined in the Linkage Section, you must have already provided addressability for identifier-3 prior to execution of the INVOKE statement. You can do this by coding either one of the following: `SET ADDRESS OF identifier-3 TO pointer` or `PROCEDURE/ENTRY USING`.

**ADDRESS OF Special Register**
See "ADDRESS OF" on page 13.

**OMITTED**
Indicates that no argument is passed.

## BY CONTENT Phrase

If BY CONTENT specified or implied for a parameter, the invoked method cannot change the value of this parameter as referenced in the INVOKE statement's USING phrase. Although, the invoked method can change the value of the data item referenced by the corresponding data-name in the invoked method's Procedure Division header. Changes to the parameter in the invoked method do not affect the corresponding argument in the invoking program.

**identifier-4**
Can be a data item of any level in the Data Division. Identifier-4 cannot be a function-identifier.

If defined in the Linkage Section, you must have already provided addressability for identifier-4 prior to execution of the INVOKE statement. You can do this by coding either one of the following: `SET ADDRESS OF identifier-4 TO pointer` or `PROCEDURE/ENTRY USING`.

**literal-2**
Can be:

- A nonnumeric literal
- A figurative constant (except ALL literal or NULL/NULLS)
- A DBCS literal

**ADDRESS OF Special Register**
See "ADDRESS OF" on page 13.

**LENGTH OF Special Register**
See "LENGTH OF" on page 15.

**OMITTED**
Indicates that no argument is passed.

## BY VALUE Phrase

The BY VALUE phrase applies to all arguments that follow until overridden by another BY REFERENCE or BY CONTENT phrase.

If the BY VALUE phrase is specified or implied for an argument, the value of the argument is passed, not a reference to the sending data item. The invoked method can modify the formal parameter corresponding to the BY VALUE argument, but any changes do not affect the argument since the invoked method has access to a temporary copy of the sending data item.

While BY VALUE arguments are primarily intended for communication with non-COBOL programs (such as C), they can also be used for COBOL-to-COBOL invocations. In this case, BY VALUE must be specified or implied for both the argument in the INVOKE USING phrase and the corresponding formal parameter in the Procedure Division USING phrase.

**identifier-5**

Must be an elementary data item in the DATA DIVISION. Identifier-5 must be one of the following:

- Binary (USAGE BINARY, COMP, COMP-4, or COMP-5)
- Floating point (USAGE COMP-1 or COMP-2)
- Pointer (USAGE POINTER)
- Procedure-pointer (USAGE PROCEDURE-POINTER)
- Object reference (USAGE OBJECT REFERENCE)
- Single-byte alphanumeric (PIC X or PIC A)

The following can also be passed BY VALUE:

- Reference modified item with length one
- SHIFT-IN and SHIFT-OUT special registers
- LINAGE-COUNTER special register when it is usage binary

**ADDRESS OF Special Register**

An ADDRESS OF special register passed BY VALUE is treated as a pointer. For information on the ADDRESS OF special register, see “ADDRESS OF” on page 13.

**LENGTH OF Special Register**

A LENGTH OF special register passed BY VALUE is treated as a PIC 9(9) binary. For information on the LENGTH OF special register, see “LENGTH OF” on page 15.

x **literal-3**
x Must be one of the following:

x - Numeric literal
x - ZERO
x - One-character nonnumeric literal
x - Symbolic character
x - Single byte figurative constant
x   – SPACE
x   – QUOTE
x   – HIGH-VALUE
x   – LOW-VALUE

x ZERO is treated as a numeric value; a fullword binary zero is passed.

x If literal-3 is a fixed point numeric literal, it must have a precision of 9 or less digits.
x In this case, a fullword binary representation of the literal value is passed.

x If literal-3 is a floating point numeric literal, an 8-byte internal floating point
x (COMP-2) representation of the value is passed.

x Literal-3 must not be a DBCS literal.

## x Conformance Requirements for USING Phrase

x The arguments specified on the USING phrase and the formal parameters specified on
x the method PROCEDURE-DIVISION USING must satisfy the following:

x - An invoked COBOL method must have the same number of formal parameters on
x its procedure division USING phrase as there are arguments on the INVOKE
x USING phrase. The presence or absence of USING...BY VALUE must be con-
x sistent on the INVOKE statement and the Procedure Division header of the target
x method.

x - If a formal parameter is a COBOL elementary data item not described with USAGE
x IS OBJECT REFERENCE, then the corresponding argument must have the same
x PICTURE, USAGE, SIGN, SYNCHRONIZED, JUSTIFIED, and BLANK WHEN
x ZERO clauses. Note that periods and commas can be interchanged if specifying
x the DECIMAL POINT IS COMMA clause and currency signs can differ.

x - If a formal parameter is a COBOL elementary data item described with USAGE IS
x OBJECT REFERENCE then:

x   – If the argument is passed BY REFERENCE, then the argument and the
x   parameter must be defined with identical USAGE clauses.

x   – If the argument is passed BY VALUE or BY CONTENT, and the parameter
x   specified on the method Procedure Division USING phrase is a universal
x   object reference, then the argument can be any object reference.

x   – If the argument is passed BY VALUE or BY CONTENT, and the parameter
x   specified on the method Procedure Division USING phrase is an object refer-
x   ence typed to a specific class, then the argument must be an object reference
x   typed to the same or a derived class.

## RETURNING Phrase

You can specify the RETURNING phrase for invoking methods written in COBOL, C, or in other programming languages that use C linkage conventions.

**identifier-6**

The RETURNING data item. Identifier-6:

- Must be defined in the DATA DIVISION
- Must not be reference-modified
- Is not changed if an EXCEPTION occurs
- Contains the return value of the invoked method

If identifier-6 is specified and the target method is written in COBOL, then the target method must have a RETURNING phrase on its Procedure Division statement. When the target returns, its return value is assigned to identifier-6, using the rules for the SET statement if identifier-6 is USAGE IS INDEX, USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE; otherwise, the rules for the MOVE statement are used.

**Note:** Identifier-6 is an output-only parameter. The initial state of identifier-6 has an unpredictable value when the method is entered. You must initialize the Procedure Division RETURNING data-name (data-name-2) in the method before you manipulate the value of identifier-6. The value that is passed back to the invoking program is the final value of identifier-6 when the method returns.

The RETURN-CODE special register is not set by execution of INVOKE statements.

***Conformance Requirements for RETURNING Phrase:*** The identifiers specified on an INVOKE RETURNING phrase and the corresponding method Procedure Division RETURNING phrase must satisfy the following:

- The presence or absence of the RETURNING phrase must be consistent on the INVOKE statement and the Procedure Division header of the target method.
- If one of the identifiers is a COBOL elementary data item not described with USAGE IS OBJECT REFERENCE, then the corresponding identifier must have the same PICTURE, USAGE, SIGN, SYNCHRONIZED, JUSTIFIED, and BLANK WHEN ZERO clauses. Note that periods and commas can be interchanged if specifying the DECIMAL POINT IS COMMA clause and currency signs can differ.
- If the INVOKE RETURNING identifier is a universal object reference, then the method Procedure Division RETURNING identifier must be an object reference (either a universal object reference or an object reference typed to a specific class).
- If the INVOKE RETURNING identifier is an object reference typed to a specific class, then the method Procedure Division RETURNING identifier must be an object reference typed to the same class or a derived class.

## ON EXCEPTION Phrase

An exception condition occurs when invoked methods are not supported by the method identified by literal-1 or identifier-2. When an exception condition occurs, one of the following two actions occur:

1. If the ON EXCEPTION phrase **is** specified, control is transferred to imperative-statement-1.
2. If the ON EXCEPTION phrase **is not** specified, then a condition is raised at run time.

Exceptions can occur if conformance requirements are not met. Conformance requirements include:

- For the USING phrase, see "Conformance Requirements for USING Phrase" on page 323
- For the RETURNING phrase, see "Conformance Requirements for RETURNING Phrase" on page 324
- For the purpose of conformance checking, a fixed-length group data item is considered to be equivalent to an elementary alphanumeric data item of the same length.

  A variable-length group conforms only to other variable-length groups that have the same maximum length.

For an example, see "INVOKE Parameter Type Conformance—Example" on page 326.

## NOT ON EXCEPTION Phrase

If an exception condition does not occur (that is, the invoked method is supported by the specified object), control is transferred to the invoked method. After control is returned from the invoked method, control is then transferred:

1. To imperative-statement-2, if the NOT ON EXCEPTION phrase **is** specified.
2. To the end of the INVOKE statement if the NOT ON EXCEPTION phrase **is not** specified.

## END-INVOKE Phrase

This explicit scope terminator serves to delimit the scope of the INVOKE statement. END-INVOKE permits a conditional INVOKE statement to be nested in another conditional statement.

**Note:** The RETURN-CODE special register is not set by execution of INVOKE statements.

## INVOKE Parameter Type Conformance—Example

```
WORKING-STORAGE SECTION.
01 anA USAGE OBJECT REFERENCE A.
01 aB  USAGE OBJECT REFERENCE B.
01 aC  USAGE OBJECT REFERENCE C.
  ⋮
PROCEDURE DIVISION
INVOKE anX "METHOD-1" USING BY REFERENCE anA.
                            BY VALUE     aB.
                       RETURNING         aC.
CLASS-ID. X.
  ⋮
METHOD-ID. METHOD-1.
  ⋮
LINKAGE SECTION.
01 aP  USAGE OBJECT REFERENCE P.
01 aQ  USAGE OBJECT REFERENCE Q.
01 anR USAGE OBJECT REFERENCE R.
  ⋮
PROCEDURE DIVISION USING BY REFERENCE aP
                         BY VALUE     aQ
                  RETURNING         anR.
```

In the above examples:

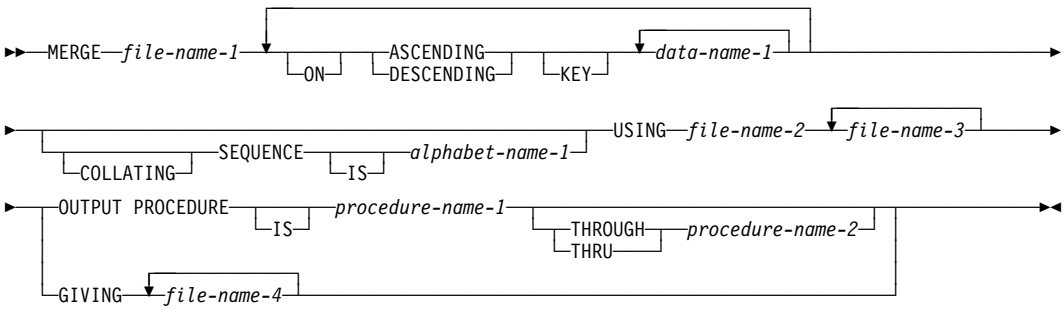
- Class P and class A must be the same class.
- Class Q must be the same class or a parent of B.
- Class R must be the same class or a subclass of C.

## MERGE Statement

The MERGE statement combines two or more identically sequenced files (that is, files that have already been sorted according to an identical set of ascending/descending keys) on one or more keys and makes records available in merged order to an output procedure or output file.

A MERGE statement can appear anywhere in the Procedure Division except in a Declarative Section.

**Format**




**file-name-1**

The name given in the SD entry that describes the records to be merged.

No file-name can be repeated in the MERGE statement.

No pair of file-names in a MERGE statement can be specified in the same SAME AREA, SAME SORT AREA, or SAME SORT-MERGE AREA clause.

x As an IBM extension, any file-names in a MERGE statement can be specified in
x the same SAME RECORD AREA clause.

When the MERGE statement is executed, all records contained in file-name-2, file-name-3,..., are accepted by the merge program and then merged according to the key(s) specified.

## ASCENDING/DESCENDING KEY Phrase

This phrase specifies that records are to be processed in an ascending or descending sequence (depending on the phrase specified), based on the specified merge keys.

**data-name-1**

Specifies a KEY data item on which the merge will be based. Each such data-name must identify a data item in a record associated with **file-name-1**. 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 left-most data-name is the major key, the next data-name is the next most significant key, and so forth.

The following rules apply:

- A specific key data item must be physically located in the same position and have the same data format in each input file. However, it need not have the same data-name.
- If file-name-1 has more than one record description, then the KEY data items need be described in only one of the record descriptions.
- If file-name-1 contains variable-length records, all of the KEY data-items must be contained within the first *n* character positions of the record, where *n* equals the minimum records size specified for file-name-1.
- KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.
- KEY data items can be qualified.
- KEY data items cannot be group items that contain variable occurrence data items.
- x KEY data items can be floating-point items.
- x KEY data items cannot be variably-located.

The direction of the merge operation depends on the specification of the ASCENDING or DESCENDING key words as follows:

- When ASCENDING is specified, the sequence is from the lowest key value to the highest key value.
- When DESCENDING is specified, the sequence is from the highest key value to the lowest.
- If the KEY data item is alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited, the sequence of key values depends on the collating sequence used (see “COLLATING SEQUENCE Phrase” on page 329 below).
- x MVS/VM Under MVS and VM, if the KEY is a DBCS item, the sequence of x the KEY values are based on the binary collating sequence of the hexadecimal x values of the DBCS characters.
- x WRKSTN Under AIX, OS/2, and Windows, if the KEY is a DBCS item, the x sequence of the KEY values are based on the collating sequence as specified

by the locale. For more information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 543.

- If the KEY is an external floating-point item, the key is treated as alphanumeric. The sequence in which the records are merged depends on the collating sequence used.
- If the KEY is an internal floating-point item, the sequence of key values will be in numeric order.

The key comparisons are performed according to the rules for comparison of operands in a relation condition (see "Relation Condition" on page 229).

## COLLATING SEQUENCE Phrase

This phrase specifies the collating sequence to be used in nonnumeric comparisons for the KEY data items in this merge operation.

WRKSTN Under AIX, OS/2, and Windows, the COLLATING SEQUENCE phrase is only valid when an ASCII code page is in effect.

**alphabet-name-1**
Must be specified in the ALPHABET clause of the SPECIAL-NAMES paragraph. Any one of the alphabet-name clause phrases can be specified, with the following results:

**STANDARD-1**

MVS/VM Under MVS and VM, the ASCII collating sequence is used for all nonnumeric comparisons. (The ASCII collating sequence is in Appendix B, "EBCDIC and ASCII Collating Sequences" on page 522.)

WRKSTN Under AIX, OS/2, and Windows, the collating sequence is based on the character's hex value order.

**STANDARD-2**

MVS/VM Under MVS and VM, the International Reference Version of the ISO 7-bit code defined in International Standard 646, 7-bit Coded Character Set for Information Processing Interchange is used for all nonnumeric comparisons.

WRKSTN Under AIX, OS/2, and Windows the collating sequence is based on the character's hex value order.

**NATIVE**

MVS/VM Under MVS and VM, the EBCDIC collating sequence is used for all nonnumeric comparisons. (The EBCDIC collating sequence is in Appendix B, "EBCDIC and ASCII Collating Sequences" on page 522.)

WRKSTN Under AIX, OS/2, and Windows, the collating sequence indicated by the locale is selected.

## MERGE Statement

**EBCDIC**
The EBCDIC collating sequence is used for all nonnumeric comparisons. (The EBCDIC collating sequence is in Appendix B, "EBCDIC and ASCII Collating Sequences" on page 522.)

**literal**
The collating sequence established by the specification of literals in the alphabet-name clause is used for all nonnumeric comparisons.

When the COLLATING SEQUENCE phrase is omitted, the PROGRAM COLLATING SEQUENCE clause (if specified) in the OBJECT-COMPUTER paragraph specifies the collating sequence to be used. When both the COLLATING SEQUENCE phrase and the PROGRAM COLLATING SEQUENCE clause are omitted,

- MVS/VM Under MVS and VM, the EBCDIC collating sequence is used. (See Appendix B, "EBCDIC and ASCII Collating Sequences" on page 522.)
- WRKSTN Under AIX, OS2, and Windows, the COLLSEQ compiler option indicates the collating sequence used. If COLLSEQ(EBCDIC) is specified, the EBCDIC collating sequence is used. If COLLSEQ(NATIVE) is specified, the collating sequence as indicated by the locale is used. For more information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 543.

## USING Phrase

**file-name-2, file-name-3, ...**
Specifies the input files.

During the MERGE operation, all the records on file-name-2, file-name-3, ... (that is, the input files) are transferred to file-name-1. At the time the MERGE statement is executed, these files must not be open. The input files are automatically opened, read, and closed, and if DECLARATIVE procedures are specified for these files for input operations, the files will be driven for errors if errors occur.

All input files must specify sequential or dynamic access mode and be described in FD entries in the Data Division.

If file-name-1 contains variable-length records, the size of the records contained in the input files (file-name-2, file-name-3, ...) must not be less than the smallest record nor greater than the largest record described for file-name-1. If file-name-1 contains fixed-length records, the size of the records contained in the input files must not be greater than the largest record described for file-name-1. For more information, see the *IBM COBOL Programming Guide* for your platform.

## GIVING Phrase

**file-name-4, ...**
Specifies the output files.

When the GIVING phrase is specified, all the merged records in file-name-1 are automatically transferred to the output files (file-name-4...).

All output files must specify sequential or dynamic access mode and be described in FD entries in the DATA DIVISION.

If the output files (file-name-4,...) contain variable-length records, the size of the records contained in file-name-1 must not be less than the smallest record nor greater than the largest record described for the output files. If the output files contain fixed-length records, the size of the records contained in file-name-1 must not be greater than the largest record described for the output files. For more information, see the *IBM COBOL Programming Guide* for your platform.

At the time the MERGE statement is executed, the output files (file-name-4,...) must not be open. The output files are automatically opened, read, and closed, and if DECLARATIVE procedures are specified for these files for output operations, the files will be driven for errors if errors occur.

## OUTPUT PROCEDURE Phrase

This phrase specifies the name of a procedure that is to select or modify output records from the merge operation.

**proc-name-1**
Specifies the first (or only) section or paragraph in the OUTPUT PROCEDURE.

**proc-name-2**
Identifies the last section or paragraph of the OUTPUT PROCEDURE.

The OUTPUT PROCEDURE can consist of any procedure needed to select, modify, or copy the records that are made available one at 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 by CALL, EXIT, GO TO, and PERFORM statements in the range of the output procedure. The range also includes 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 must not cause the execution of any MERGE, RELEASE, or SORT statement.

If an output procedure is specified, control passes to it after the file referenced by file-name-1 has been sequenced by the MERGE statement. The compiler inserts a return mechanism at the end of the last statement in the output procedure and when control passes the last statement in the output procedure, the return mechanism provides the 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 can select the next record in merged order when requested, The RETURN statements in the output procedure are the requests for the next record.

**Note:** The OUTPUT PROCEDURE phrase is similar to a basic PERFORM statement. For example, if you name a procedure in an OUTPUT PROCEDURE, that procedure is executed during the merging operation just as if it were named in a PERFORM statement. As with the PERFORM statement, execution of the procedure is terminated after the last statement completes execution. The last statement in an OUTPUT PROCEDURE can be the EXIT statement (see “EXIT Statement” on page 299).

## MERGE Statement

## MERGE Special Registers

**SORT-CONTROL Special Register**

You define the sort control file (through which you can specify additional options to the sort/merge function) with the SORT-CONTROL special register.

If you use a sort control file to specify control statements, the values specified in the sort control file take precedence over those in the special register.

For information, see “SORT-CONTROL” on page 18.

**SORT-MESSAGE Special Register (MVS and VM Only)**

For information, see “SORT-MESSAGE” on page 20.

MVS/VM The special register SORT-MESSAGE is equivalent to an option control statement key word in the sort control file.

**SORT-RETURN Special Register**

For information, see “SORT-RETURN” on page 21.

## Segmentation Considerations

If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear:

1. Totally within non-independent segments, or
2. Wholly contained in a single independent segment.

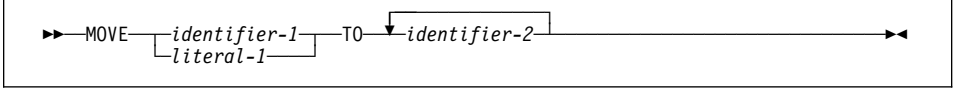
If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

1. Totally within non-independent segments, or
2. Wholly within the same independent segment as that MERGE statement.

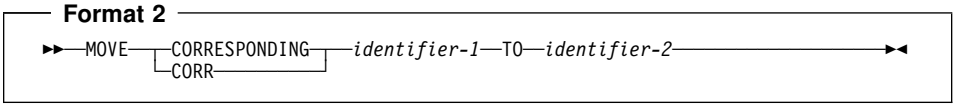
## MOVE Statement

The MOVE statement transfers data from one area of storage to one or more other areas.

**Format 1**




**Format 2**




**identifier-1, literal-1**
Sending area

**identifier-2**
Receiving area(s)

When Format 1 is specified, all identifiers can be either group or elementary items. The data in the sending area is moved into the data item referenced by each identifier-2 in the order in which it is specified. See “Elementary Moves” on page 334 and “Group Moves” on page 337.

When Format 2 is specified, both identifiers must be group items. CORR is an abbreviation for, and is equivalent to, CORRESPONDING.

When CORRESPONDING is specified, selected items in identifier-1 are moved to identifier-2, according to the rules for the CORRESPONDING phrase on page 249. The results are the same as if each pair of CORRESPONDING identifiers were referenced in a separate MOVE statement.

x Do not specify a data item defined with USAGE IS POINTER, USAGE IS
x PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE in a MOVE statement.

x A data item defined with USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or
x USAGE IS OBJECT REFERENCE can be part of a group that is referred to in a MOVE
x CORRESPONDING statement; however, no movement of the data item will take place.

An index data item cannot be specified in a MOVE statement.

The evaluation of the length of the sending or receiving area can be affected by the DEPENDING ON phrase of the OCCURS clause (see “OCCURS Clause” on page 163).

If the sending field (identifier-1) is reference-modified, subscripted, or is an alphanumeric or alphabetic function-identifier, the reference-modifier, subscript, or function is

evaluated only once, immediately before data is moved to the first of the receiving operands.

Any length evaluation, subscripting, or reference-modification associated with a receiving field (identifier-2) is evaluated immediately before the data is moved into that receiving field.

For example, 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 defined as an intermediate result item. The subscript `B` has changed in value between the time that the first move took place and the time that the final move to `C(B)` is executed.

For further information on intermediate results, see the *IBM COBOL Programming Guide* for your platform.

After execution of a MOVE statement, the sending field(s) contain the same data as before execution.

**Note:** Overlapping operands in a MOVE statement can cause unpredictable results.

## Elementary Moves

An elementary move is one in which the receiving item is an elementary item, and the sending item is an elementary item or a literal. Any necessary conversion of data from one form of internal representation to another takes place during the move, along with any specified editing in, or de-editing implied by, the receiving item. Each elementary item belongs to one of the following categories:

**Alphabetic**—includes alphabetic data items and the figurative constant SPACE.

**Alphanumeric**—includes alphanumeric data items, nonnumeric literals, and all figurative constants except SPACE and ZERO (when ZERO is moved to a numeric or numeric-edited item).

**Alphanumeric-Edited**—includes alphanumeric-edited data items.

**Numeric**—includes numeric data items, numeric literals, and the figurative constant ZERO (when ZERO is moved to a numeric or numeric-edited item).

**Numeric-Edited**—includes numeric-edited data items.

x **Floating-point**—includes internal floating-point items (defined as USAGE COMP-1
x or USAGE COMP-2), external floating-point items (defined as USAGE DISPLAY),
x and floating-point literals.

x **DBCS**—includes DBCS data items (defined explicitly or implicitly as USAGE
x DISPLAY-1) and DBCS literals.

The following rules outline the execution of valid elementary moves. When the receiving field is:

**Alphabetic**:

- Alignment and any necessary space filling occur as described under "Alignment Rules" on page 137.
- If the size of the sending item is greater than the size of the receiving item, excess characters on the right are truncated after the receiving item is filled.

**Alphanumeric** or **Alphanumeric-Edited**:

- Alignment and any necessary space filling take place, as described under "Alignment Rules" on page 137.
- If the size of the sending item is greater than the size of the receiving item, excess characters on the right are truncated after the receiving item is filled.
- If the sending item has an operational sign, the unsigned value is used. If the operational sign occupies a separate character, that character is not moved, and the size of the sending item is considered to be one less character than the actual size.

**Numeric** or **Numeric-edited**:

- Except where zeros are replaced because of editing requirements, alignment by decimal point and any necessary zero filling take place, as described under "Alignment Rules" on page 137.
- If the receiving item is signed, the sign of the sending item is placed in the receiving item, with any necessary sign conversion. If the sending item is unsigned, a positive operational sign is generated for the receiving item.
- If the receiving item is unsigned, the absolute value of the sending item is moved, and no operational sign is generated for the receiving item.
- When the sending item is alphanumeric, the data is moved as if the sending item were described as an unsigned integer.
- x When the sending item is floating-point, the data is first converted to either a binary x or internal decimal representation and is then moved.
- De-editing allows moving a numeric-edited data item into a numeric or numeric-edited receiver. The compiler accomplishes this by first establishing the unedited value of the numeric-edited item (this value can be signed), then moving the unedited numeric value to the receiving numeric or numeric-edited data item.

x **Floating-point**:

- x The sending item is converted first to internal floating-point and then moved.
- x When data is moved to or from an external floating-point item, the data is con- x verted first to or from its equivalent internal floating-point value.

**DBCS:**

- No conversion takes place.
- If the sending and receiving items are not the same size, the data item will be either truncated or padded with DBCS spaces on the right.

  WRKSTN Under AIX, OS/2, and Windows, if the padding required is not in a multiple consistent with double-byte characters, single-byte characters will be used (for example, a group item moved to a DBCS data item).

**Notes:**

1. If the receiving field is alphanumeric or numeric-edited, and the sending field is a scaled integer (that is, has a P as the rightmost character in its PICTURE character-string), the scaling positions are treated as trailing zeros when the MOVE statement is executed.
2. If the receiving field is numeric and the sending field is alphanumeric literal or ALL literal, then all characters of the literal must be numeric characters.

Table 38 shows valid and invalid elementary moves for each category. In the figure:

- YES = Move is valid.
- NO = Move is invalid.

*Table 38. Valid and Invalid Elementary Moves*

| | Receiving Item Category | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Sending Item Category** | **Alphabetic** | **Alphanumeric** | **Alphanumeric Edited** | **Numeric** | **Numeric Edited** | **External Floating Point** | **Internal Floating Point** | **DBCS[1]** |
| **Alphabetic and SPACE** | Yes | Yes | Yes | No | No | No | No | No |
| **Alphanumeric[2]** | Yes | Yes | Yes | Yes[3] | Yes[3] | Yes[8] | Yes[8] | No |
| **Alphanumeric-Edited** | Yes | Yes | Yes | No | No | No | No | No |
| **Numeric Integer and ZERO[4]** | No | Yes | Yes | Yes | Yes | Yes | Yes | No |
| **Numeric Non-integer[5]** | No | No | No | Yes | Yes | Yes | Yes | No |
| **Numeric-Edited** | No | Yes | Yes | Yes | Yes | Yes | Yes | No |
| **Floating-point[6]** | No | No | No | Yes | Yes | Yes | Yes | No |
| **DBCS[7]** | No | No | No | No | No | No | No | Yes |

**Note:**

1 Includes DBCS data items.
2 Includes nonnumeric literals.
3 Figurative constants and nonnumeric literals must consist only of numeric characters and will be treated as numeric integer fields.
4 Includes integer numeric literals.
5 Includes non-integer numeric literals.
6 Includes floating-point literals, external floating-point data items (USAGE DISPLAY), and internal floating-point data items (USAGE COMP-1 or USAGE COMP-2).
7 Includes DBCS data-items, DBCS literals, and SPACE.
8 Figurative constants and nonnumeric literals must consist only of numeric characters and will be treated as numeric integer fields. The ALL literal cannot be used as a sending item.

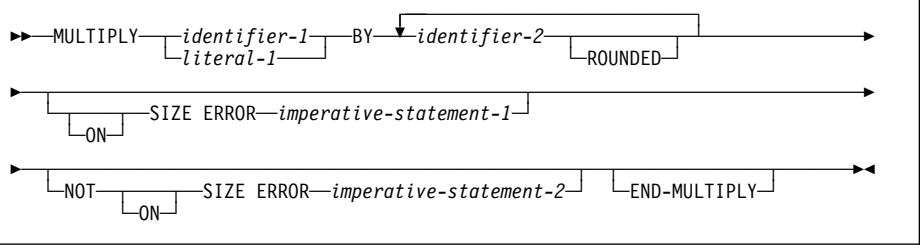
## Group Moves

A group move is one in which one or both of the sending and receiving fields are group items. A group move is treated exactly as though it were an alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In a group move, the receiving area is filled without consideration for the individual elementary items contained within either the sending area or the receiving area, except as noted in the OCCURS clause. (See “OCCURS Clause” on page 163.)
**All** group moves are valid.

## MULTIPLY Statement

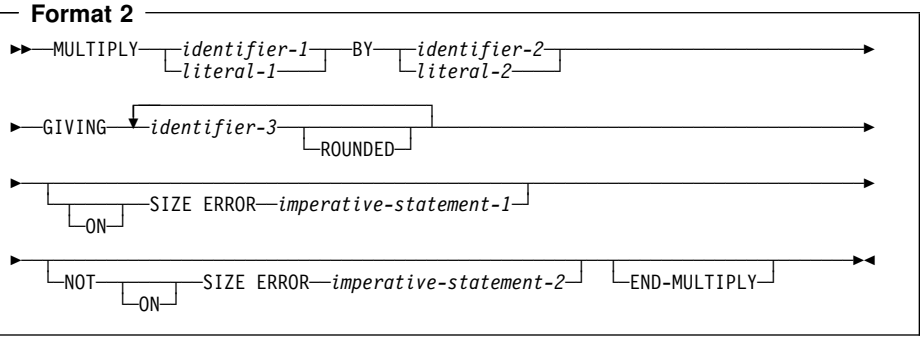
The MULTIPLY statement multiplies numeric items and sets the values of data items equal to the results.

**Format 1**




In Format 1, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2; the product is then placed in identifier-2. For each successive occurrence of identifier-2, the multiplication takes place in the left-to-right order in which identifier-2 is specified.

**Format 2**




In Format 2, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2. The product is then stored in the data item(s) referenced by identifier-3.

***For all Formats:***

**identifier-1, identifier-2**
Must name an elementary numeric item.

**literal-1, literal-2**
Must be a numeric literal.

***For Format-2:***

**identifier-3**
Must name an elementary numeric or numeric-edited item.

x Floating-point data items and literals can be used anywhere a numeric data item or
x literal can be specified.

The composite of operands must not contain more than 18 digits.

x The composite of operands can be more than 18 digits. For information on arithmetic
x intermediate results, see the *IBM COBOL Programming Guide* for your platform.

## ROUNDED Phrase

For Formats 1 and 2, see “ROUNDED Phrase” on page 250.

## SIZE ERROR Phrases

For Formats 1 and 2, see “SIZE ERROR Phrases” on page 251.

## END-MULTIPLY Phrase

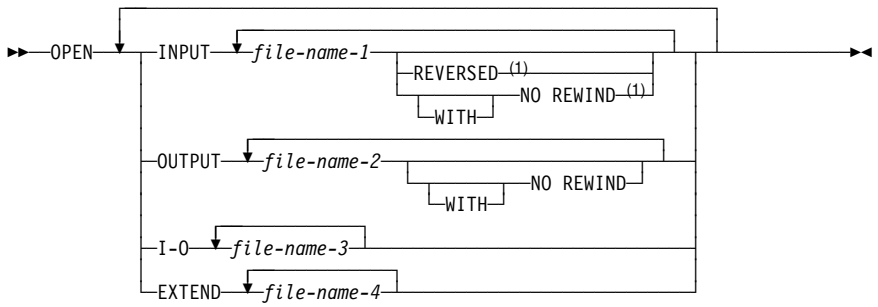
This explicit scope terminator serves to delimit the scope of the MULTIPLY statement. END-MULTIPLY permits a conditional MULTIPLY statement to be nested in another conditional statement. END-MULTIPLY can also be used with an imperative MULTIPLY statement.

For more information, see “Delimited Scope Statements” on page 247.

## OPEN Statement

The OPEN statement initiates the processing of files. It also checks and/or writes labels.

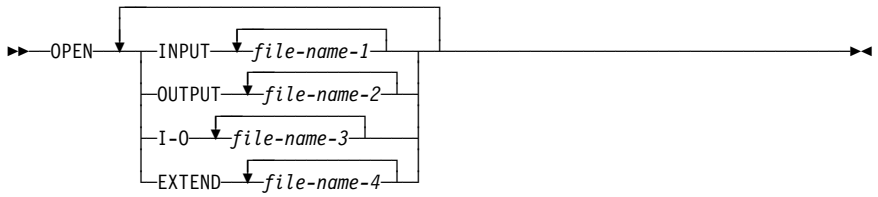
**Format 1—Sequential Files**



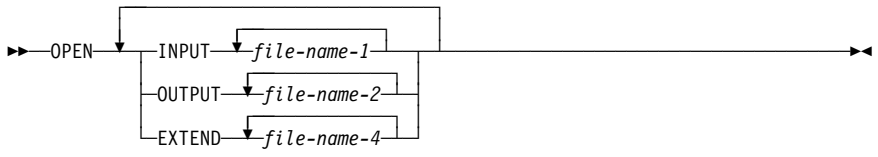

**Note:**

1 Under MVS, the REVERSED and WITH NO REWIND phrases are not valid for VSAM files. Under AIX, OS/2, and Windows, the REVERSED and WITH NO REWIND phases are treated as a comment.

**Format 2—Indexed and Relative Files**




**Format 3—Line Sequential Files (Workstation Only)**




At least one of the phrases, INPUT, OUTPUT, I-O, or EXTEND, must be specified with the OPEN key word. The INPUT, OUTPUT, I-O, and EXTEND phrases can appear in any order.

**INPUT**
Permits opening the file for input operations.

**OUTPUT**
Permits opening the file for output operations. This phrase can be specified when the file is being created.

**Note:** Do not specify OUTPUT for files that:

- Contain records. The file will be replaced by new data. If the OUTPUT phrase is specified for a file that already contains records, the data set must be defined as reusable and cannot have an alternate index. The records in the file will be replaced by the new data and any ALTERNATE RECORD KEY clause in the SELECT statement will be ignored.
- Under MVS, are defined with a DD dummy card. Unpredictable results can occur.

**I-O** Permits opening the file for both input and output operations. The I-O phrase can be specified only for files assigned to direct access devices.

x WRKSTN Under AIX, OS/2, and Windows, the I-O phrase is not valid for line
x sequential files.

**EXTEND**
Permits opening the file for output operations.

The EXTEND phrase is only allowed for sequential access files if the new data is
x written in ascending sequence. As an IBM extension, the EXTEND phrase is
x allowed for files for that specify the LINAGE clause.

Under MVS, for QSAM files, do not specify the EXTEND phrase for a multiple file reel.

**file-name-1, file-name-2, file-name-3, file-name-4**
Designates a file upon which the OPEN statement is to operate. If more than one file is specified, the files need not have the same organization or access. Each file-name must be defined in an FD entry in the Data Division, and must not name a sort or merge file. The FD entry must be equivalent to the information supplied when the file was defined.

**REVERSED**
MVS/VM Under MVS and VM, the REVERSED phrase is only valid for sequential single reel files. It is not valid for VSAM files.

If the concept of reels has no meaning for the storage medium (for example, a direct access device), the REVERSED and NO REWIND phrases do not apply.

WRKSTN Under AIX, OS/2, and Windows, the REVERSED phrase is treated as a comment.

**NO REWIND**

MVS/VM Under MVS and VM, the NO REWIND phrase is only valid for sequential single reel files. It is not valid for VSAM files.

WRKSTN Under AIX, OS/2, and Windows, the NO REWIND phrase is treated as a comment.

## General Rules

- If a file opened with the INPUT phrase is an optional file which is not present, the OPEN statement sets the file position indicator to indicate that an optional input file is not present.
- Execution of an OPEN INPUT or OPEN I-O statement sets the file position indicator:
  - For indexed files, to the characters with the lowest ordinal position in the collating sequence associated with the file.
  - For sequential and relative files, to 1.
- When the EXTEND phrase is specified, the OPEN statement positions the file immediately after the last record written in the file. (The record with the highest prime record key value (for indexed files) or relative key value (for relative files) is considered the last record.) Subsequent WRITE statements add records as if the file were opened OUTPUT. The EXTEND option can be specified when a file is being created; it can also be specified for a file that contains records, or that has contained records that have been deleted.
- WRKSTN Under AIX, OS/2, and Windows, any files with the LOCK MODE IS AUTOMATIC phrase (if specified) are shareable. Shareable files can be opened by more than one file connector. If the OPEN statement fails due to locking constraints, the file status is set to indicate “file locked.”
- When the EXTEND phrase is not specified, OPEN statement positions the file at its beginning.
- MVS/VM For MVS VSAM files, if no records exist in the file, the file position indicator is set so that the first Format 1 READ statement executed results in an AT END condition.
- When NO REWIND is specified, the OPEN statement execution does not reposition the file; prior to OPEN statement execution, the file must be positioned at its beginning. When the NO REWIND phrase is specified (or when both the NO REWIND and REVERSE phrases are omitted), file positioning is specified with the LABEL parameter of the DD statement under MVS and with the "label processing" operand under CMS.
- When REVERSED is specified, OPEN statement execution positions the QSAM file at its end. Subsequent READ statements make the data records available in reversed order, starting with the last record.

  When OPEN REVERSED is specified, the record format must be fixed.

- When the REVERSED, NO REWIND, or EXTEND phrases are not specified, OPEN statement execution positions the file at its beginning. ◁MT

x MVS/VM⇨ If the PASSWORD clause is specified in the FILE-CONTROL entry, the pass-
x word data item must contain the valid password before the OPEN statement is exe-
x cuted. If the valid password is not present, the OPEN statement execution is
x unsuccessful. ◁MT

## Label Records

WRKSTN⇨ Under AIX, OS/2, and Windows, label processing is not supported. A warning message is issued if any of the following language elements are encountered:

- LABEL RECORDS IS data-name
- USE...AFTER...LABEL PROCEDURE
- GO TO MORE-LABELS ◁

If label records are specified for the file when the OPEN statement is executed, the labels are processed according to the standard label conventions, as follows:

**INPUT files** The beginning labels are checked.

**OUTPUT files** The beginning labels are written.

**I-O files** The labels are checked; new labels are then written.

**EXTEND files** The following procedures are executed:

- Beginning file labels are processed only if this is a single-volume file.
- Beginning volume labels of the last existing volume are processed as though the file was being opened with the INPUT phrase.
- Existing ending file labels are processed as though the file was being opened with the INPUT phrase; they are then deleted.
- Processing continues as if the file were opened as an OUTPUT file.

When label records are specified but not present, or are present but not specified, execution of the OPEN statement is unpredictable.

## OPEN Statement Notes

1. The successful execution of an OPEN statement determines the availability of the file and results in that file being in open mode. A file is available if it is physically present and is recognized by the input-output control system. Table 39 on page 344 shows the results of opening available and unavailable files. For more information regarding file availability, see the *IBM COBOL Programming Guide* for your platform.

*Table 39. Availability of a File*

| OPENed As | 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 the 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 |

2. The successful execution of the OPEN statement makes the associated record area available to the program; it does not obtain or release the first data record.
3. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements, except a SORT or MERGE statement with the USING or GIVING phrase. Table 40 shows the permissible input-output statements for sequential files. An 'X' indicates that the specified statement can be used with the open mode given at the top of the column.

*Table 40. Permissible Statements for Sequential Files*

| Statement | Open Mode | | | |
|---|---|---|---|---|
| | Input | Output | I-O | Extend |
| **READ** | X | | X | |
| **WRITE** | | X | | X |
| **REWRITE** | | | X | |

Table 41 on page 345 shows the permissible statements for indexed and relative files. An 'X' indicates that the specified statement, used in the access mode given for that row, can be used with the OPEN mode given at the top of the column.

*Table 41. Permissible Statements for Indexed and Relative Files*

| File Access Mode | Statement | Open Mode | | | |
|---|---|---|---|---|---|
| | | Input | Output | I-O | Extend |
| Sequential | READ | X | | X | |
| | WRITE | | X | | X |
| | REWRITE | | | X | |
| | START | X | | X | |
| | 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 | |

Table 42 shows the permissible input-output statements for line sequential files. An 'X' indicates that the specified statement can be used with the open mode given at the top of the column.

*Table 42. Permissible Statements for Line Sequential Files (Workstation Only)*

| Statement | Open Mode | | | |
|---|---|---|---|---|
| | **Input** | **Output** | **I-O** | **Extend** |
| **READ** | X | | | |
| **WRITE** | | X | | X |
| **REWRITE** | | | | |

4. A file can be opened for INPUT, OUTPUT, I-O, or EXTEND (sequential and line sequential files only) in the same program. After the first OPEN statement execution for a given file, each subsequent OPEN statement execution must be preceded by a successful CLOSE file statement execution without the REEL or UNIT phrase (for QSAM files only), or the LOCK phrase.
5. If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the OPEN statement is executed.
6. If an OPEN statement is issued for a file already in the open status, the EXCEPTION/ERROR procedure (if specified) for this file is executed.

## PERFORM Statement

The PERFORM statement transfers control explicitly to one or more procedures and implicitly returns control to the next executable statement after execution of the specified procedure(s) is completed.

The PERFORM statement can be:

**An out-of-line PERFORM statement**
Procedure-name-1 is specified.

**An in-line PERFORM statement**
Procedure-name-1 is omitted.

An in-line PERFORM must be delimited by the END-PERFORM phrase.

The in-line and out-of-line formats cannot be combined. For example, if procedure-name-1 is specified, the imperative-statement and the END-PERFORM phrase must not be specified.

The PERFORM statement formats are:

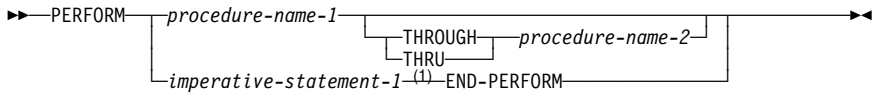
- Basic PERFORM
- TIMES phrase PERFORM
- UNTIL phrase PERFORM
- VARYING phrase PERFORM

## Basic PERFORM Statement

The procedure(s) referenced in the basic PERFORM statement are executed once, and control then passes to the next executable statement following the PERFORM statement.

**Note:** A PERFORM statement must not cause itself to be executed. Such a recursive PERFORM statement can cause unpredictable results.

**Format 1**




**Note:**
x [1] Imperative-statement-1 is optional as an IBM extension.

**procedure-name-1, procedure-name-2**
Must name a section or paragraph in the Procedure Division.

When both procedure-name-1 and procedure-name-2 are specified, if either is a procedure-name in a declarative procedure, both must be procedure-names in the same declarative procedure.

If procedure-name-1 is specified, imperative-statement-1 and the END-PERFORM phrase must not be specified.

If procedure-name-1 is omitted, imperative-statement and the END-PERFORM phrase must be specified.

**imperative-statement**
The statements to be executed for an in-line PERFORM.

An in-line PERFORM statement functions according to the same general rules as an otherwise identical out-of-line PERFORM statement, except that statements contained within the in-line PERFORM are executed in place of the statements contained within the range of procedure-name-1 (through procedure-name-2, if specified). Unless specifically qualified by the word **in-line** or **out-of-line**, all the rules that apply to the out-of-line PERFORM statement also apply to the in-line PERFORM.

Whenever an out-of-line PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. Control is always returned to the statement following the PERFORM statement. The point from which this control is returned is determined as follows:

- If procedure-name-1 is a paragraph name and procedure-name-2 is not specified, the return is made after the execution of the last statement of the procedure-name-1 paragraph.
- If procedure-name-1 is a section name and procedure-name-2 is not specified, the return is made after the execution of the last statement of the last paragraph in the procedure-name-1 section.
- If procedure-name-2 is specified and it is a paragraph name, the return is made after the execution of the last statement of the procedure-name-2 paragraph.
- If procedure-name-2 is specified and it is a section name, the return is made after the execution of the last statement of the last paragraph in the procedure-name-2 section.

The only necessary relationship between procedure-name-1 and procedure-name-2 is that a consecutive sequence of operations is executed, beginning at the procedure named by procedure-name-1 and ending with the execution of the procedure named by procedure-name-2.

PERFORM statements can be specified within the performed procedure. If there are two or more logical paths to the return point, then procedure-name-2 can name a paragraph that consists only of an EXIT statement; all the paths to the return point must then lead to this paragraph.

When the performed procedures include another PERFORM statement, the sequence of procedures associated with the embedded PERFORM statement must be totally included in or totally excluded from the performed procedures of the first PERFORM statement. That is, an active PERFORM statement whose execution point begins within the range of performed procedures of another active PERFORM statement must not allow control to pass through the exit point of the other active PERFORM state-

## PERFORM Statement

ment. As an IBM extension, two or more active PERFORM statements can have a common exit.

Figure 11 illustrates valid sequences of execution for PERFORM statements.

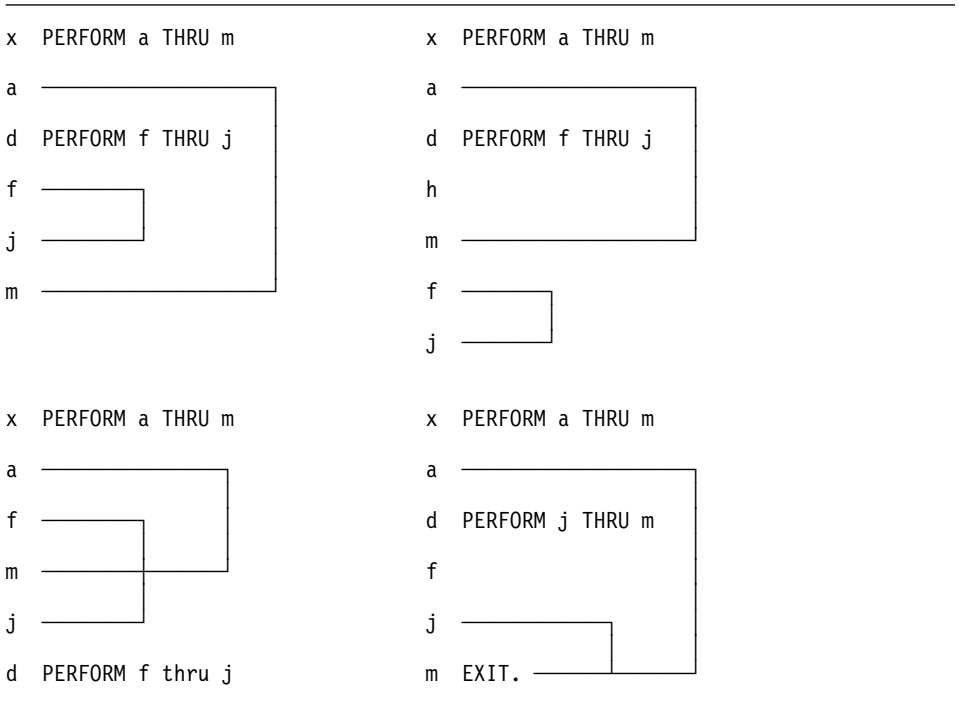



*Figure 11. Valid PERFORM Statement Execution Sequences*

When control passes to the sequence of procedures by means other than a PERFORM statement, control passes through the exit point to the next executable statement, as if no PERFORM statement referred to these procedures.

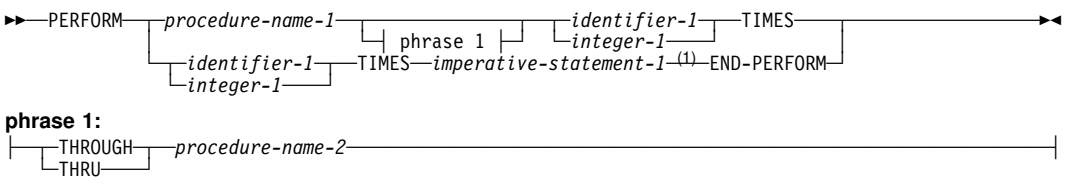
## END-PERFORM

Delimits the scope of the in-line PERFORM statement. Execution of an in-line PERFORM is completed after the last statement contained within it has been executed.

## PERFORM with TIMES Phrase

The procedure(s) referred to in the TIMES phrase PERFORM statement are executed the **number of times** specified by the value in identifier-1 or integer-1. Control then passes to the next executable statement following the PERFORM statement.

**Format 2**




**Note:**

x [1] Imperative-statement-1 is optional as an IBM extension.

**Note:** If procedure-name-1 is specified, imperative-statement and the END-PERFORM phrase must not be specified.

**identifier-1**

Must name an integer item.

If identifier-1 is zero or a negative number at the time the PERFORM statement is initiated, control passes to the statement following the PERFORM statement.

After the PERFORM statement has been initiated, any change to identifier-1 has no effect in varying the number of times the procedures are initiated.

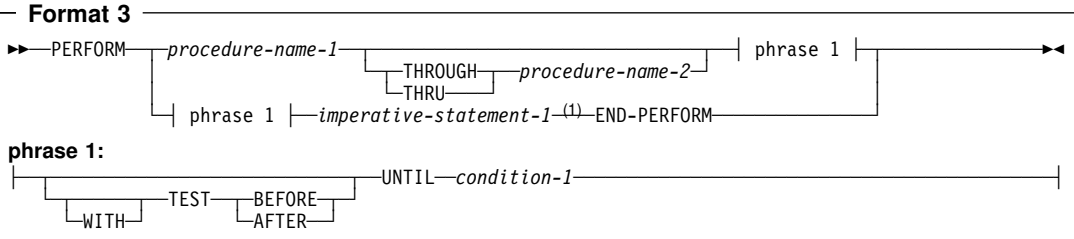
**integer-1**

x Can be a positive signed integer.

## PERFORM with UNTIL Phrase

In the UNTIL phrase format, the procedure(s) referred to are performed **until** the condition specified by the UNTIL phrase is true. Control is then passed to the next executable statement following the PERFORM statement.

**Format 3**




**Note:**

x [1] Imperative-statement-1 is optional as an IBM extension.

**Note:** If procedure-name-1 is specified, imperative-statement-1 and the END-PERFORM phrase must not be specified.

**condition-1**

Can be any condition described under “Conditional Expressions” on page 225. If the condition is true at the time the PERFORM statement is initiated, the specified procedure(s) are not executed.

Any subscripting associated with the operands specified in condition-1 is evaluated each time the condition is tested.

If the TEST BEFORE phrase is specified or assumed, the condition is tested before any statements are executed (corresponds to DO WHILE).

If the TEST AFTER phrase is specified, the statements to be performed are executed at least once before the condition is tested (corresponds to DO UNTIL).

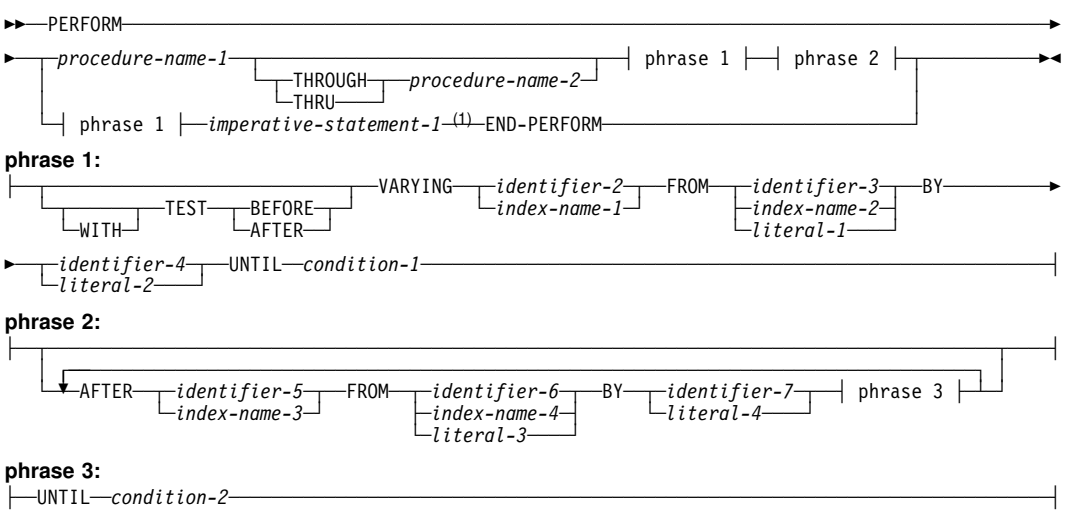
In either case, if the condition is true, control is transferred to the next executable statement following the end of the PERFORM statement. If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.

## PERFORM with VARYING Phrase

The VARYING phrase increases or decreases the value of one or more identifiers or index-names, according to certain rules. (See "Varying Phrase Rules" on page 356.)

The Format 4 VARYING phrase PERFORM statement can serially search an entire 7-dimensional table.

**Format 4**




**Note:**

x 1 Imperative-statement-1 is optional as an IBM extension.

**Note:** If procedure-name-1 is specified, imperative-statement and the END-PERFORM phrase must not be specified. If procedure-name-1 is omitted, the AFTER phrase must not be specified.

**identifier-2 thru 7**
Must name a numeric elementary item.

**literal-1 thru 4**
Must represent a numeric literal

**condition-1, condition-2**
Can be any condition described under "Conditional Expressions" on page 225. If the condition is true at the time the PERFORM statement is initiated, the specified procedure(s) are not executed.

After the condition(s) specified in the UNTIL phrase are satisfied, control is passed to the next executable statement following the PERFORM statement.

If any of the operands specified in condition-1 or condition-2 is subscripted, reference modified, or is a function-identifier, the subscript, reference-modifier, or function is evaluated each time the condition is tested.

x Floating-point data items and literals can be used anywhere a numeric data item or
x literal can be specified.

When TEST BEFORE is indicated, all specified conditions are tested before the first execution, and the statements to be performed are executed, if at all, only when **all** specified tests fail. When TEST AFTER is indicated, the statements to be performed are executed at least once, before any condition is tested.

If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.

## Varying Identifiers

The way in which operands are increased or decreased depends on the number of variables specified. In the following discussion, every reference to identifier-n refers equally to index-name-n (except when identifier-n is the object of the BY phrase).

If identifier-2 or identifier-5 is subscripted, the subscripts are evaluated each time the content of the data item referenced by the identifier is set or augmented. If identifier-3, identifier-4, identifier-6, or identifier-7 is subscripted, the subscripts are evaluated each time the content of the date item referenced by the identifier is used in a setting or an augmenting operation.

Figure 12 on page 352 illustrates the logic of the PERFORM statement when an identifier is varied with TEST BEFORE.

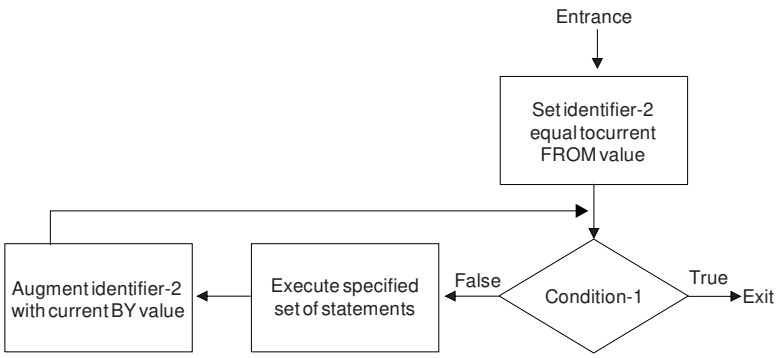



*Figure 12. Varying One Identifier—with TEST BEFORE*

Figure 13 illustrates the logic of the PERFORM statement when an identifier is varied with TEST AFTER.

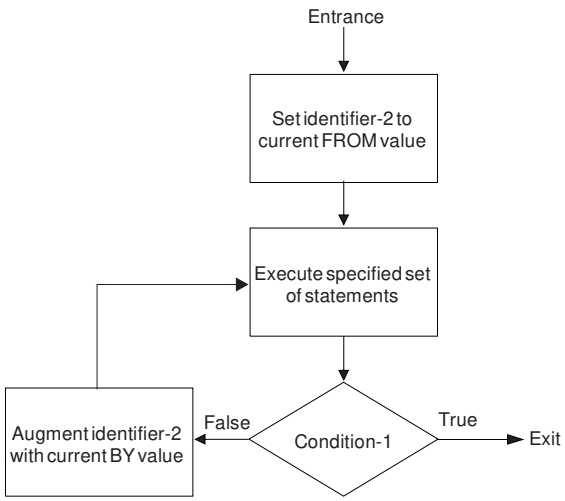



*Figure 13. Varying One Identifier—with TEST AFTER*

## Varying Two Identifiers

```
PERFORM PROCEDURE-NAME-1 THROUGH PROCEDURE-NAME-2
      VARYING IDENTIFIER-2 FROM IDENTIFIER-3
            BY IDENTIFIER-4 UNTIL CONDITION-1
         AFTER IDENTIFIER-5 FROM IDENTIFIER-6
            BY IDENTIFIER-7 UNTIL CONDITION-2
```

1. **identifier-2** and **identifier-5** are set to their initial values, identifier-3 and identifier-6, respectively.
2. **condition-1** is evaluated as follows:
   a. If it is false, steps 3 through 7 are executed.
   b. If it is true, control passes directly to the statement following the PERFORM statement.
3. **condition-2** is evaluated as follows:
   a. If it is false, steps 4 through 6 are executed.
   b. If it is true, identifier-2 is augmented by identifier-4, identifier-5 is set to the current value of identifier-6, and step 2 is repeated.
4. **procedure-1** and **procedure-2** are executed once (if specified).
5. **identifier-5** is augmented by identifier-7.
6. Steps 3 through 5 are repeated until condition-2 is true.
7. Steps 2 through 6 are repeated until condition-1 is true.

At the end of PERFORM statement execution:

- **identifier-5**

  Contains the current value of identifier-6.

- **identifier-2**

  Has a value that exceeds the last-used setting by the increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case, identifier-2 contains the current value of identifier-3).

Figure 14 illustrates the logic of the PERFORM statement when two identifiers are varied with TEST BEFORE.

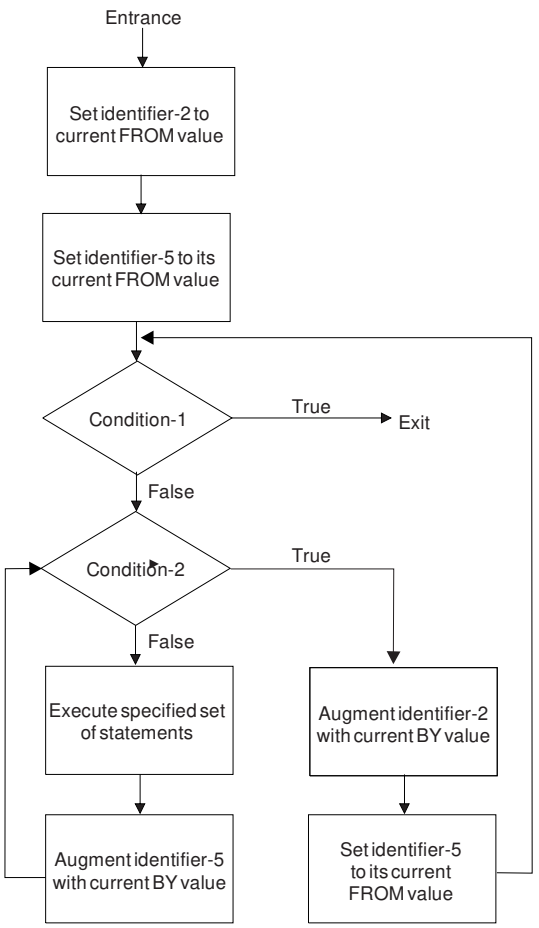



*Figure 14. Varying Two Identifiers—with TEST BEFORE*

Figure 15 illustrates the logic of the PERFORM statement when two identifiers are varied with TEST AFTER.

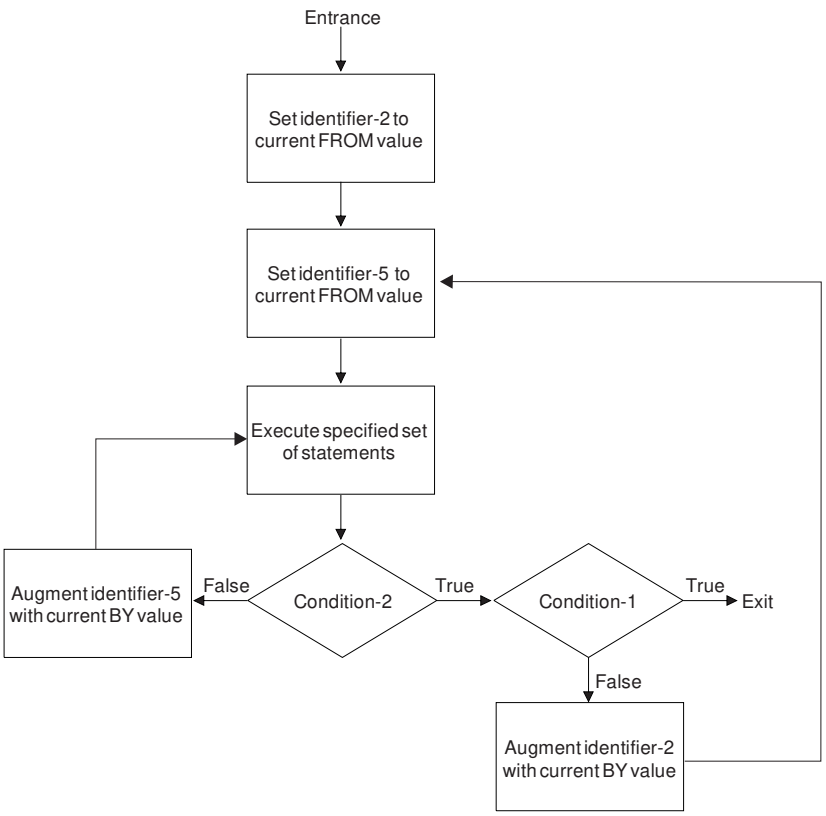



*Figure 15. Varying Two Identifiers—with TEST AFTER*

## Varying Three Identifiers

```
PERFORM PROCEDURE-NAME-1 THROUGH PROCEDURE-NAME-2
      VARYING IDENTIFIER-2 FROM IDENTIFIER-3
            BY IDENTIFIER-4 UNTIL CONDITION-1
          AFTER IDENTIFIER-5 FROM IDENTIFIER-6
            BY IDENTIFIER-7 UNTIL CONDITION-2
          AFTER IDENTIFIER-8 FROM IDENTIFIER-9
            BY IDENTIFIER-10 UNTIL CONDITION-3
```

The actions are the same as those for two identifiers, except that identifier-8 goes through the complete cycle each time identifier-5 is augmented by identifier-7, which, in turn, goes through a complete cycle each time identifier-2 is varied.

At the end of PERFORM statement execution:

- **identifier-5** and **identifier-8**

  Contain the current values of identifier-6 and identifier-9, respectively.

- **identifier-2**

  Has a value exceeding its last-used setting by one increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case, identifier-2 contains the current value of identifier-3).

## Varying More Than Three Identifiers

You can produce analogous PERFORM statement actions to the example above with the addition of up to four AFTER phrases.

## Varying Phrase Rules

No matter how many variables are specified, the following rules apply:

1. In the VARYING/AFTER phrases, when an index-name is specified:
   a. The index-name is initialized and incremented or decremented according to the rules under “INDEXED BY Phrase” on page 166. (See also “SET Statement” on page 381.)
   b. In the associated FROM phrase, an identifier must be described as an integer and have a positive value; a literal must be a positive integer.
   c. In the associated BY phrase, an identifier must be described as an integer; a literal must be a nonzero integer.
2. In the FROM phrase, when an index-name is specified:
   a. In the associated VARYING/AFTER phrase, an identifier must be described as an integer. It is initialized, as described in the SET statement.
   b. In the associated BY phrase, an identifier must be described as an integer and have a nonzero value; a literal must be a nonzero integer.
3. In the BY phrase, identifiers and literals must have nonzero values.

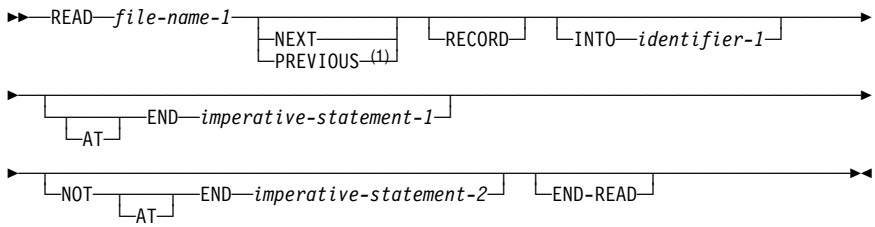
4. Changing the values of identifiers and/or index-names in the VARYING, FROM, and BY phrases during execution changes the number of times the procedures are executed.

## READ Statement

For sequential access, the READ statement makes the next logical record from a file available to the object program. For random access, the READ statement makes a specified record from a direct-access file available to the object program.

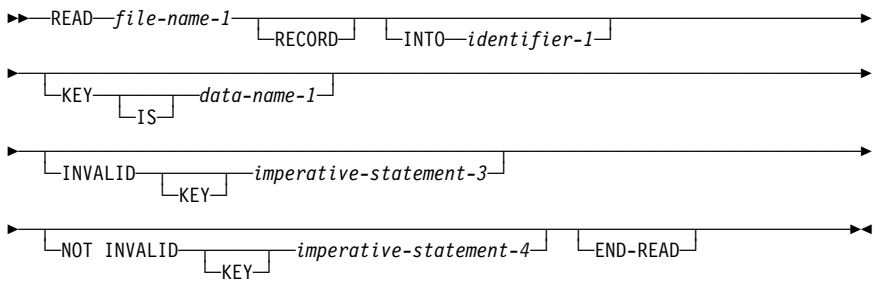
When the READ statement is executed, the associated file must be open in INPUT or I-O mode.

**Format 1—Sequential Retrieval**




**Note:**
1 PREVIOUS is only supported on AIX, OS/2, and Windows.

**Format 2—Random Retrieval**




**file-name-1**
Must be defined in a Data Division FD entry.

**NEXT RECORD**
The next record in the logical sequence of records. NEXT is optional when ACCESS MODE IS SEQUENTIAL; it has no effect on READ statement execution.

MVS/VM Under MVS and VM, you must specify the NEXT RECORD phrase for files in dynamic access mode, which are retrieved sequentially.

WRKSTN Under AIX, OS/2, and Windows, you must specify either the NEXT or PREVIOUS phrase for files in dynamic access mode, which are retrieved sequentially.

**PREVIOUS RECORD (Workstation Only)**

WRKSTN The previous record in the logical sequence of records. PREVIOUS applies to indexed and relative files with DYNAMIC access mode.

You must specify either the NEXT or PREVIOUS phrase for files in dynamic access mode, which are retrieved sequentially.

If you specify READ...PREVIOUS and no previous logical record exists, the AT END condition occurs and the READ statement is unsuccessful.

When you specify READ...PREVIOUS, the setting of the file position indicator is used to determine which record to make available according to the following rules:

- If the file position indicator indicates that no valid previous record has been established, the READ is unsuccessful.
- If the file position indicator is positioned by the execution of an OPEN statement, the AT END condition occurs.
- If the file position indicator is established by a previous START statement, the first existing record in the file whose relative record number (if a relative file) or whose key value (if an indexed file) is less than or equal to the file position indicator is selected.
- If the file position indicator is established by a previous READ statement, the first existing record in the file whose relative record number (if a relative file) or whose key value (if an indexed file) is less than the file position indicator is selected.

**INTO Identifier-1**

Identifier-1 is the receiving field.

The result of the execution of a READ statement with the INTO phrase is equivalent to the application of the following rules in the order specified:

- The execution of the same READ statement without the INTO phrase.
- 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 a group move. The implied MOVE statement does not occur if the execution of the READ statement was unsuccessful. Any subscripting or reference modification associated with identifier-1 is evaluated 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.

The INTO phrase can be specified in a READ if:

- Only one record description is subordinate to the file description entry, or
- All record-names associated with file-name-1, and the data item referenced by identifier-1, describe a group item or an elementary alphanumeric item.

The record areas associated with file-name-1 and identifier-1 must not be the same storage area.

x Identifier-1 (the record area) can be a DBCS data item.

x Identifier-1 can be a floating-point data item.

x Multiple non-alphanumeric records can be specified for file-name-1. Identifier-1
x need not describe a group item or an elementary alphanumeric item. The following
x rules apply:

x 1. If the file referenced by file-name-1 is V-format, or a QSAM file with
x RECORDING MODE 'S' or 'U', a group move will take place.

x 2. If the file referenced by file-name-1 is F-format, the movement will take place
x according to the rules for the MOVE statement, using, as a sending field
x description, the record that specifies the largest number of character positions.
x If more than one such record exists, the sending field record selected will be
x the one among those records that appears first under the description of
x file-name-1.

## KEY IS Phrase

The KEY IS phrase can be specified only for indexed files. Data-name-1 must identify a record key associated with file-name-1. Data-name-1 can be qualified; it may not be subscripted.

x **Data-name-1** (the record key) can be defined as a DBCS data item.

x When the RECORD KEY clause specifies a DBCS data item, a KEY specified on the
x READ statement must be a DBCS data item.

## AT END Phrases

For sequential access, the AT END phrase must be specified if no applicable USE AFTER STANDARD EXCEPTION procedure is specified for file-name-1.

x The AT END phrase does not have to be specified if no applicable USE AFTER
x STANDARD EXCEPTION procedure is specified for file-name-1.

For information on at-end condition processing, see “At End Condition” on page 362.

## INVALID KEY Phrases

For random access, the INVALID KEY phrase must be specified if no applicable USE AFTER STANDARD EXCEPTION procedure is specified for file-name-1.

x The INVALID KEY phrase does not have to be specified if no applicable USE AFTER
x STANDARD EXCEPTION procedure is specified for file-name-1.

For information on INVALID KEY phrases processing, see “Invalid Key Condition” on page 258.

## END-READ Phrase

This explicit scope terminator serves to delimit the scope of the READ statement. END-READ permits a conditional READ statement to be nested in another conditional statement. END-READ can also be used with an imperative READ statement. For more information, see “Delimited Scope Statements” on page 247.

## Multiple Record Processing

If more than one record description entry is associated with file-name-1, these records automatically share the same storage area; that is, they are implicitly redefined. After a READ statement is executed, only those data items within the range of the current record are replaced; data items stored beyond that range are undefined. Figure 16 illustrates this concept. If the range of the current record exceeds 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 (04) is set indicating a record length conflict has occurred.

---

The FD entry is:
FD INPUT-FILE LABEL RECORDS OMITTED.

01 RECORD-1 PICTURE X(30).

01 RECORD-2 PICTURE X(20).

Contents of input area when READ statement is executed:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234
```

Contents of record being read in (RECORD-2):

```
01234567890123456789
```

Contents of input area after READ is executed:

```
01234567890123456789??????????
```

(these characters in input area are undefined)

---

*Figure 16. READ Statement with Multiple Record Description*

## Sequential Access Mode

Format 1 must be used for all files in sequential access mode.

Execution of a Format 1 READ statement retrieves the next logical record from the file. The next record accessed is determined by the file organization.

### Sequential Files

The NEXT RECORD is the next record in a logical sequence of records. The NEXT phrase need not be specified; it has no effect on READ statement execution.

If SELECT OPTIONAL is specified in the FILE-CONTROL entry for this file, and the file is absent during this execution of the object program, execution of the first READ statement causes an at end condition; however, since no file is present, the system-defined end-of-file processing is not performed.

***At End Condition:*** If the file position indicator indicates that no next logical record exists, or that an optional input file is not present, the following occurs in the order specified:

1. A value, derived from the setting of the file position indicator, is placed into the I-O status associated with file-name-1 to indicate the at end condition.
2. If the AT END phrase is specified in the statement causing the condition, control is transferred to imperative-statement-1 in the AT END phrase. Any USE AFTER STANDARD EXCEPTION procedure associated with file-name-1 is not executed.
3. If the AT END phrase is not specified, a USE AFTER STANDARD EXCEPTION procedure must be specified with this file, and the procedure is executed. Return from that procedure is to the next executable statement following the end of the READ statement.

   x If the AT END phrase is not specified, a USE AFTER STANDARD EXCEPTION
   x procedure does not have to be associated with file-name-1.

   When the at end condition occurs, execution of the READ statement is unsuccessful. The contents of the associated record area are undefined and the file position indicator is set to indicate that no valid next record has been established. Attempts to access or move data into the read record area following an unsuccessful read can result in a protection exception.

If an at end condition does not occur during the execution of a READ statement, the AT END phrase is ignored, if specified, and the following actions occur:

1. The file position indicator is set and the I-O status associated with file-name-1 is updated.
2. If an exception condition that is not an at end condition exists, control is transferred to the end of the READ statement following the execution of any USE AFTER STANDARD EXCEPTION procedure applicable to file-name-1.

   If no USE AFTER STANDARD EXCEPTION procedure is specified, control is transferred to the end of the READ statement or to imperative-statement-2, if specified.

3. 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 to imperative-statement-2, if specified. In the latter case, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement which 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.

Following the unsuccessful execution of a READ statement, the contents of the associated record area are undefined and the file position indicator is set to indicate that no valid next record has been established. Attempts to access or move data into the record area following an unsuccessful read can result in a protection exception.

***Multivolume QSAM Files (MVS and VM Only):*** If end-of-volume is recognized during execution of a READ statement, and logical end-of-file has not been reached, the following actions are taken:

- The standard ending volume label procedure
- A volume switch
- The standard beginning volume label procedure
- The first data record of the next volume is made available.

## Indexed or Relative Files

The NEXT RECORD is the next logical record in the key sequence.

x WRKSTN Under AIX, OS/2, and Windows, PREVIOUS RECORD is the prior logical
x record in the key sequence.

For indexed files, the key sequence is the sequence of ascending values of the current key of reference. For relative files, the key sequence is the sequence of ascending values of relative record numbers for records that exist in the file.

Before the READ statement is executed, the file position indicator must be set by a successful OPEN, START, or READ statement. When the READ statement is executed, the record indicated by the file position indicator is made available, if it is still accessible through the path indicated by the file position indicator.

If the record is no longer accessible (because it has been deleted, for example), the file
x position indicator is updated to point to the next (or previous) existing record in the file, and that record is made available.

For files in sequential access mode, the NEXT phrase need not be specified.

x For files in dynamic access mode, the NEXT phrase (or alternatively for AIX, OS/2, and
x Windows, the PREVIOUS phrase) must be specified for sequential record retrieval.

## READ Statement

***AT END Condition:*** If the file position indicator indicates that no next logical record exists, (or alternatively for AIX, OS/2, and Windows, no previous record exits), or that an optional input file is not present.

MVS/VM Under MVS and VM, the same procedure occurs as for sequential files (see "At End Condition" on page 362).

WRKSTN Under AIX, OS/2, and Windows see the discussion on PREVIOUS RECORD on 358.

If neither an at end nor an invalid key condition occurs during the execution of a READ statement, the AT END or the INVALID KEY phrase is ignored, if specified. The same actions occur as when the at end condition does not occur with sequential files (see "At End Condition" on page 362).

***Sequentially Accessed Indexed Files:*** When an ALTERNATE RECORD KEY with DUPLICATES is the key of reference, file records with duplicate key values are made available in the order in which they were placed in the file.

***Sequentially Accessed Relative Files:*** If the RELATIVE KEY clause is specified for this file, READ statement execution updates the RELATIVE KEY data item to indicate the relative record number of the record being made available.

### Line Sequential Files

WRKSTN The LOCK clause is not valid for line sequential files that are opened for INPUT.

You can only READ a file (with implicit lock) if no other file connector holds a lock on the record you are trying to access. If the record is locked by another file connector, the READ statement is unsuccessful and the file status is set to indicate "record locked." For sequential files, the setting of the file position indicator is unaffected.

When no next record exist when a READ statement is executed, the AT END condition is returned regardless of any sharing of the file (for example, if the file is opened in EXTEND mode by another file connector).

## Random Access Mode

Format 2 must be specified for indexed and relative files in random access mode, and also for files in the dynamic access mode when record retrieval is random.

WRKSTN Under AIX, OS/2, and Windows, for line sequential files, if a file connector holds a LOCK on a file in random access mode, the READ statement is unsuccessful. The setting of the file position indicator is unspecified.

Execution of the READ statement depends on the file organization, as explained in the following sections.

### Indexed Files

Execution of a Format 2 READ statement causes the value of the key of reference to be compared with the value of the corresponding key data item in the file records, until the first record having an equal value is found. The file position indicator is positioned to this record, which is then made available. If no record can be so identified, an INVALID KEY condition exists, and READ statement execution is unsuccessful. (See "INVALID KEY Condition" under "Common Processing Facilities" on page 254.)

If the KEY phrase is not specified, the prime RECORD KEY becomes the key of reference for this request. When dynamic access is specified, the prime RECORD KEY is also used as the key of reference for subsequent executions of sequential READ statements, until a different key of reference is established.

When the KEY phrase is specified, data-name becomes the key of reference for this request. When dynamic access is specified, this key of reference is used for subsequent executions of sequential READ statements, until a different key of reference is established.

### Relative Files

Execution of a Format 2 READ statement sets the file position indicator pointer to the record whose relative record number is contained in the RELATIVE KEY data item, and makes that record available.

If the file does not contain such a record, the INVALID KEY condition exists, and READ statement execution is unsuccessful. (See "Invalid Key Condition" under "Common Processing Facilities" on page 254.)

The KEY phrase must not be specified for relative files.

## Dynamic Access Mode

For files with indexed or relative organization, dynamic access mode can be specified in the FILE-CONTROL entry. In dynamic access mode, either sequential or random record retrieval can be used, depending on the format used.

Format 1 with the NEXT phrase must be specified for sequential retrieval. All other rules for sequential access apply.

## READ Statement Notes

If the FILE-STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the READ statement is executed.

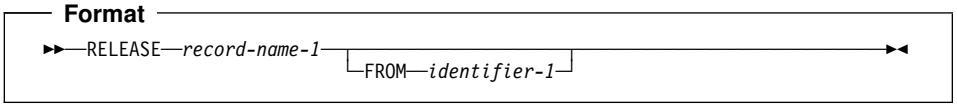
Following unsuccessful READ statement execution, the contents of the associated record area and the value of the file position indicator are undefined. Attempts to access or move data into the record area following an unsuccessful read can result in a protection exception.

## RELEASE Statement

The RELEASE statement transfers records from an input/output area to the initial phase of a sorting operation.

The RELEASE statement can only be used within the range of an INPUT PROCEDURE associated with a SORT statement.

**Format**




Within an INPUT PROCEDURE, at least one RELEASE statement must be specified.

When the RELEASE statement is executed, the current contents of record-name-1 are placed in the sort file; that is, made available to the initial phase of the sorting operation.

**record-name-1**
Must specify the name of a logical record in a sort-merge file description entry (SD). Record-name-1 can be qualified.

x Can define a floating-point data item or DBCS data item. Identifier-1 must be a
x DBCS data item if record-name-1 is a DBCS data item.

**FROM phrase**
The result of the execution of the RELEASE statement with the FROM identifier-1 phrase is equivalent to the execution of the following statements in the order specified.

```
MOVE identifier-1 to record-name-1.
RELEASE record-name-1.
```

The MOVE is performed according to the rules for the MOVE statement without the CORRESPONDING phrase.

**identifier-1**
Identifier-1 must be one of the following:

- The name of an entry in the Working-Storage Section or the LINKAGE SECTION
- The name of a record description for another previously opened file
- The name of an alphanumeric function identifier

Identifier-1 and record-name-1 must not refer to the same storage area.

x Identifier-1 can be a floating-point data item or a DBCS data item.

After the RELEASE statement is executed, the information is still available in identifier-1. (See “INTO/FROM Identifier Phrase” under “Common Processing Facilities” on page 254.)

If the RELEASE statement is executed without specifying the SD entry for file-name-1 in a SAME RECORD AREA clause, the information in record-name-1 is no longer available.

If the SD entry **is** specified in a SAME RECORD AREA clause, record-name-1 is still available as a record of the other files named in that clause.

When FROM identifier-1 is specified, the information is still available in identifier-1.

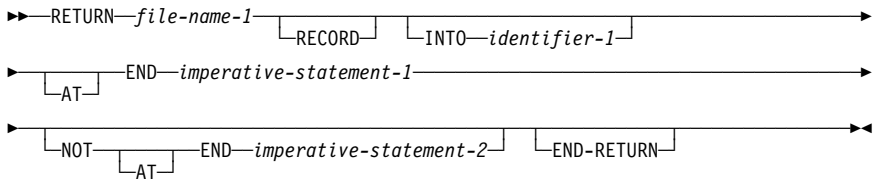
When control passes from the INPUT PROCEDURE, the sort file consists of all those records placed in it by execution of RELEASE statements.

## RETURN Statement

The RETURN statement transfers records from the final phase of a sorting or merging operation to an OUTPUT PROCEDURE.

The RETURN statement can be used only within the range of an OUTPUT PROCEDURE associated with a SORT or MERGE statement.

**Format**




Within an OUTPUT PROCEDURE, at least one RETURN statement must be specified.

When the RETURN statement is executed, the next record from file-name-1 is made available for processing by the OUTPUT PROCEDURE.

**file-name-1**

Must be described in a Data Division SD entry.

If more than one record description is associated with file-name-1, these records automatically share the same storage; that is, the area is implicitly redefined. After RETURN statement execution, only the contents of the current record are available; if any data items lie beyond the length of the current record, their contents are undefined.

**INTO phrase**

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:

- The execution of the same RETURN statement without the INTO phrase.
- 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 a group move. The implied MOVE statement does not occur if the execution of the RETURN statement was unsuccessful. Any subscripting or reference modification associated with identifier-1 is evaluated 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.

**identifier-1**

x Identifier-1 (the record area) can be a floating-point data item or DBCS data item.

The record areas associated with file-name-1 and identifier-1 must not be the same storage area.

The INTO phrase can be specified in a RETURN statement if one or both of the following are true:

- If only one record description is subordinate to the sort-merge file description entry
- If all record-names associated with file-name-1 and the data item referenced by identifier-1 describe a group item or an elementary alphanumeric item.

x Multiple non-alphanumeric records can be specified for file-name-1. Identifier-1 need
x not describe a group item or an elementary alphanumeric item. The following rules
x apply:

x 1. If the file referenced by file-name-1 contains variable-length records or, under MVS
x and VM, a QSAM file with RECORDING MODE 'S' or 'U', a group move will take
x place.

x 2. If the file referenced by file-name-1 contains fixed-length records, the movement
x will take place according to the rules for the MOVE statement, using, as a sending
x field description, the record that specifies the largest number of character positions.
x If more than one such record exists, the sending field record selected will be the
x one among those records that appears first under the description of file-name-1.

## AT END Phrases

The imperative-statement specified on the AT END phrase executes after all records have been returned from file-name-1. No more RETURN statements can be executed as part of the current output procedure.

If an at end condition does not occur 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 the imperative statement specified by the NOT AT END phrase, otherwise control is passed to the end of the RETURN statement.

## END-RETURN Phrase

This explicit scope terminator serves to delimit the scope of the RETURN statement. END-RETURN permits a conditional RETURN statement to be nested in another conditional statement. END-RETURN can also be used with an imperative RETURN statement.

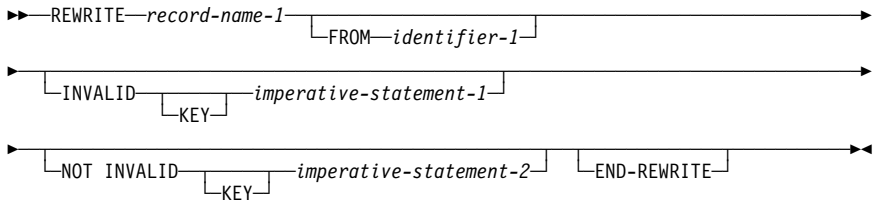
For more information, see “Delimited Scope Statements” on page 247.

## REWRITE Statement

The REWRITE statement logically replaces an existing record in a direct-access file. When the REWRITE statement is executed, the associated direct-access file must be open in I-O mode.

x WRKSTN Under AIX, OS/2, and Windows the REWRITE statement is not supported for
x line sequential files.

**Format**




**record-name-1**

Must be the name of a logical record in a Data Division FD entry. The record-name can be qualified.

x Record-name-1 can define a floating-point data item or DBCS data item.
x Identifier-1 must be a DBCS data item if record-name-1 is a DBCS data item.

**FROM phrase**

The result of the execution of the REWRITE statement with the FROM identifier-1 phrase is equivalent to the execution of the following statements in the order specified.

```
MOVE identifier-1 TO record-name-1.
REWRITE record-name-1
```

The MOVE is performed according to the rules for the MOVE statement without the CORRESPONDING phrase.

**identifier-1**

Identifier-1 must be one of the following:

- The name of an entry in the Working-Storage Section or LINKAGE SECTION
- The name of a record description for another previously opened file
- The name of an alphanumeric function identifier

Identifier-1 and record-name-1 must not refer to the same storage area.

x Identifier-1 can be a floating-point data item or a DBCS data item.

After the REWRITE statement is executed, the information is still available in identifier-1 (See “INTO/FROM Identifier Phrase” under “Common Processing Facilities” on page 254).

## INVALID KEY Phrases

(See "Invalid Key Condition" under "Common Processing Facilities" on page 254.)

An INVALID KEY condition exists when:

- The access mode is sequential, and the value contained in the prime RECORD KEY of the record to be replaced does not equal the value of the prime RECORD KEY data item of the last-retrieved record from the file, or
- The value contained in the prime RECORD KEY does not equal that of any record in the file, or
- The value of an ALTERNATE RECORD KEY data item for which DUPLICATES is not specified is equal to that of a record already in the file.

## END-REWRITE Phrase

This explicit scope terminator serves to delimit the scope of the REWRITE statement. END-REWRITE permits a conditional REWRITE statement to be nested in another conditional statement. END-REWRITE can also be used with an imperative REWRITE statement.

For more information, see "Delimited Scope Statements" on page 247.

## Affect of Record Locking (LOCK MODE Clause)

WRKSTN Under AIX, OS/2, and Windows, if any other file connector holds a LOCK on the record to be deleted, the REWRITE statement is unsuccessful. If the lock mode is AUTOMATIC, a successful REWRITE statement releases an existing record lock.

## Reusing a Logical Record

After successful execution of a REWRITE statement, the logical record is no longer available in record-name-1 unless the associated file is named in a SAME RECORD AREA clause (in which case, the record is also available as a record of the other files named in the SAME RECORD AREA clause).

The file position indicator is not affected by execution of the REWRITE statement.

If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the REWRITE statement is executed.

## Sequential Files

For files in the sequential access mode, the last prior input/output statement executed for this file must be a successfully executed READ statement. When the REWRITE statement is executed, the record retrieved by that READ statement is logically replaced.

The number of character positions in record-name-1 must equal the number of character positions in the record being replaced.

The INVALID KEY phrase must not be specified for a file with sequential organization. An EXCEPTION/ERROR procedure can be specified.

### Indexed Files

The number of character positions in record-name-1 must equal the number of character positions in the record being replaced.

x The number of character positions in record-name-1 can be different from the number
x of character positions in the record being replaced.

When the access mode is sequential, the record to be replaced is specified by the value contained in the prime RECORD KEY. When the REWRITE statement is executed, this value must equal the value of the prime record key data item in the last record read from this file.

The INVALID KEY phrase must be specified if an applicable USE AFTER STANDARD EXCEPTION procedure is not specified for the associated file-name.

x The INVALID KEY phrase does not have to be specified if an applicable USE AFTER
x STANDARD EXCEPTION procedure is not specified for the associated file-name.

When the access mode is random or dynamic, the record to be replaced is specified by the value contained in the prime RECORD KEY.

Values of ALTERNATE RECORD KEY data items in the rewritten record can differ from those in the record being replaced. The system ensures that later access to the record can be based upon any of the record keys.

If an invalid key condition exists, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, and the data in record-name-1 is unaffected. (See “INVALID KEY Condition” under “Common Processing Facilities” on page 254.)

### Relative Files

The number of character positions in record-name-1 must equal the number of character positions in the record being replaced.

x The number of character positions in record-name-1 can be different from the number
x of character positions in the record being replaced.

For relative files in sequential access mode, the INVALID KEY phrase must not be specified. An EXCEPTION/ERROR procedure can be specified.

The INVALID KEY phrase must be specified in the REWRITE statement for relative files in the random or dynamic access mode, and for which an appropriate USE AFTER STANDARD EXCEPTION procedure is not specified.

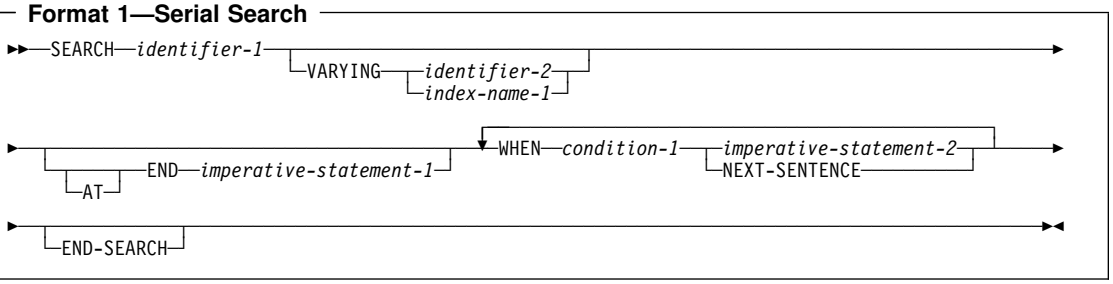
x The INVALID KEY phrase does not have to be specified if an appropriate USE AFTER
x STANDARD EXCEPTION procedure is not specified.

When the access mode is random or dynamic, the record to be replaced is specified in the RELATIVE KEY data item. If the file does not contain the record specified, an invalid key condition exists, and, if specified, the INVALID KEY imperative-statement is executed. (See “Invalid Key Condition” under “Common Processing Facilities” on page 254.) The updating operation does not take place, and the data in record-name is unaffected.

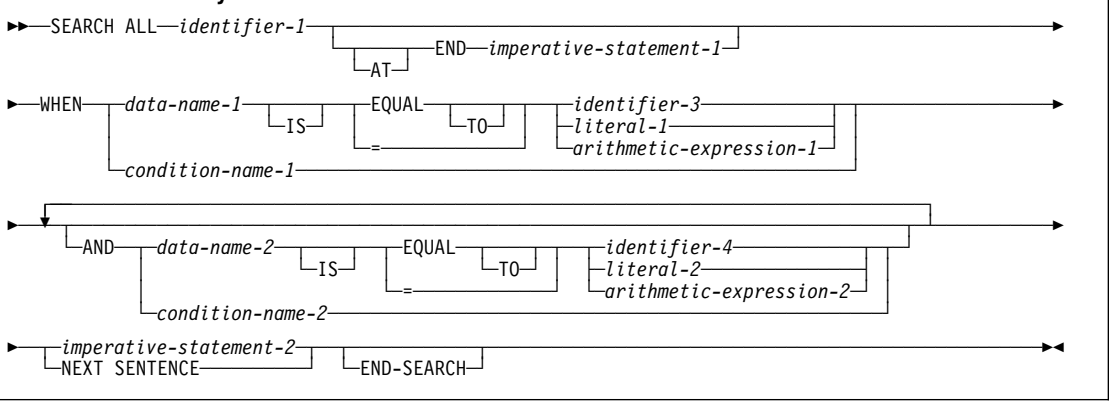
## SEARCH Statement

The SEARCH statement searches a table for an element that satisfies the specified condition, and adjusts the associated index to indicate that element.

**Format 1—Serial Search**




**Format 2—Binary Search**




**identifier-1**

Can be:

- A data item subordinate to a data item that contains an OCCURS clause; that is, it can be a part of a multidimensional table. In this case, the data description entry must specify an INDEXED BY phrase for each dimension of the table.
- An index data item
- A DBCS data item or a floating-point data item

Identifier-1 must refer to all occurrences within the table element; that is, it must not be subscripted or reference-modified.

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY phrase. For Format-2, the Data Division description must also contain the KEY IS phrase in its OCCURS clause.

SEARCH statement execution modifies only the value in the index-name associated with identifier-1 (and, if present, of index-name-1 or identifier-2). Therefore, to search an entire two- to seven-dimensional table, it is necessary to execute a

SEARCH statement for each dimension. Before each execution, SET statements must be executed to reinitialize the associated index-names.

## AT END/WHEN Phrases

After imperative-statement-1 or imperative-statement-2 is executed, control passes to the end of the SEARCH statement, unless imperative-statement-1 or imperative-statement-2 ends with a GO TO statement.

## NEXT SENTENCE

x As an IBM extension, you can specify END-SEARCH with NEXT SENTENCE. Note,
x however, that if the NEXT SENTENCE phrase is executed, control will not pass to the
x next statement following the END-SEARCH, but instead will pass to the statement after
x the closest following period.

x As an IBM extension, neither imperative-statement-2 nor NEXT SENTENCE is required.
x Without them, the SEARCH statement sets the index to the value in the table that
x matched the condition.

## END-SEARCH Phrase

This explicit scope terminator serves to delimit the scope of the SEARCH statement. END-SEARCH permits a conditional SEARCH statement to be nested in another conditional statement.

For more information, see “Delimited Scope Statements” on page 247.

## Serial Search

The Format 1 SEARCH statement executes a serial search beginning at the current index setting. When the search begins, if the value of the index-name associated with identifier-1 is not greater than the highest possible occurrence number, the following actions take place:

- The condition(s) in the WHEN phrase are evaluated in the order in which they are written.
- If none of the conditions is satisfied, the index-name for identifier-1 is increased to correspond to the next table element, and step 1 is repeated.
- If upon evaluation, one of the WHEN conditions is satisfied, the search is terminated immediately, and the imperative-statement associated with that condition is executed. The index-name points to the table element that satisfied the condition. If NEXT SENTENCE is specified, control passes to the statement following the closest period.
- If the end of the table is reached (that is, the incremented index-name value is greater than the highest possible occurrence number) without the WHEN condition being satisfied, the search is terminated, as described in the next paragraph.

If, when the search begins, the value of the index-name associated with identifier-1 is greater than the highest possible occurrence number, the search immediately ends, and, if specified, the AT END imperative-statement is executed. If the AT END phrase is omitted, control passes to the next statement after the SEARCH statement.

## VARYING Phrase

**index-name-1**

One of the following actions applies:

- If index-name-1 is an index for identifier-1, this index is used for the search. Otherwise, the first (or only) index-name is used.
- If index-name-1 is an index for another table element, then the first (or only) index-name for identifier-1 is used for the search; the occurrence number represented by index-name-1 is increased by the same amount as the search index-name and at the same time.

When the VARYING index-name-1 phrase is omitted, the first (or only) index-name for identifier-1 is used for the search.

x If indexing is used to search a table without an INDEXED BY clause, correct
x results are ensured only if both the table defined with the index and the table
x defined without the index have table elements of the same length and with the
x same number of occurrences.

**identifier-2**

Must be either an index data item or an elementary integer item. Identifier-2 cannot be subscripted by the first (or only) index-name for identifier-1. During the search, one of the following actions applies:

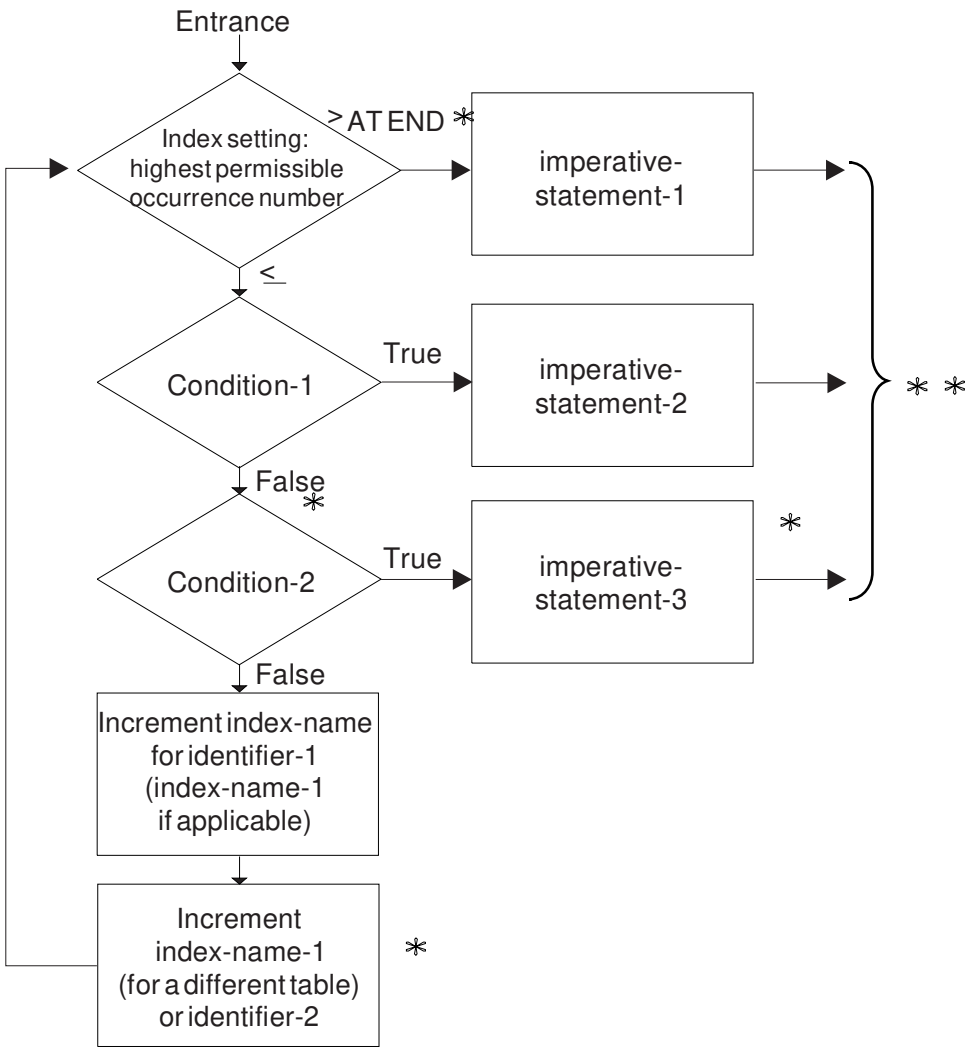
- If identifier-2 is an index data item, then, whenever the search index is increased, the specified index data item is simultaneously increased by the same amount.
- If identifier-2 is an integer data item, then, whenever the search index is increased, the specified data item is simultaneously increased by 1.

## WHEN Phrase (Serial Search)

**condition-1**
**condition-2**

Can be any condition described under “Conditional Expressions” on page 225.

Figure 17 illustrates a Format 1 SEARCH operation containing two WHEN phrases.




* These operations are included only when called for in the statement.
* * Control transfers to the next sentence, unless the imperative statement ends with a GO TO statement.

*Figure 17. Format 1 SEARCH with Two WHEN Phrases*

## Binary Search

The Format 2 SEARCH ALL statement executes a binary search. The search index need not be initialized by SET statements, because its setting is varied during the search operation so that its value is at no time less than the value of the first table element, nor ever greater than the value of the last table element. The index used is always that associated with the first index-name specified in the OCCURS clause.

The results of a SEARCH ALL operation are predictable **only** when:

- The data in the table is ordered in ASCENDING/DESCENDING KEY
- The contents of the ASCENDING/DESCENDING keys specified in the WHEN clause provide a unique table reference.

**identifier-1**

Identifier-1 **can** be:

- A data item subordinate to a data item that contains an OCCURS clause; that is, it can be a part of a two- to seven-dimensional table. In this case, the data description entry must specify an INDEXED BY phrase for each dimension of the table.
- A DBCS item if the ASCENDING/DESCENDING KEY is defined as a DBCS item.

Identifier-1 **cannot** be:

- USAGE IS INDEX
- A floating-point data item
- A data item defined with USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE

Identifier-1 must refer to all occurrences within the table element; that is, it must not be subscripted or reference-modified.

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY option. It must also contain the KEY IS phrase in its OCCURS clause.

**AT END**

The condition that exists when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

## WHEN Phrase (Binary Search)

If the WHEN relation-condition is specified, the compare is based on the length and sign of data-name. For example, if the length of data-name is shorter than the length of the search argument, the search argument is truncated to the length of data-name before the compare is done. If the search argument is signed and data-name is unsigned, the sign is removed from the search argument before the compare is done.

If the WHEN phrase **cannot** be satisfied for any setting of the index within this range, the search is unsuccessful. Control is passed to imperative-statement-1 of the AT END phrase, when specified, or to the next statement after the SEARCH statement. In either case, the final setting of the index is not predictable.

If the WHEN option **can** be satisfied, control passes to imperative-statement-2, if specified, or to the next executable sentence if the NEXT SENTENCE phrase is specified. The index contains the value indicating the occurrence that allowed the WHEN condition(s) to be satisfied.

**condition-name**
Each condition-name specified must have only a single value, and each must be associated with an ASCENDING/DESCENDING KEY identifier for this table element.

**data-name**
Must specify an ASCENDING/DESCENDING KEY data item in the identifier-1 table element and must be subscripted by the first identifier-1 index-name. Each data-name can be qualified.

x Cannot be a floating-point data item.

Cannot be a group item containing variable occurrence data items.

**identifier-3**
**identifier-4**
Must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is subscripted by the first index-name for identifier-1.

x Can be floating-point data items.

x Cannot be data items defined with the USAGE IS POINTER, USAGE IS
x PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE.

**arithmetic-expression**
Can be any of the expressions defined under “Arithmetic Expressions” on page 223, with the following restriction: Any identifier in the arithmetic-expression must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is subscripted by the first index-name for identifier-1.

When an ASCENDING/DESCENDING KEY data item is specified, explicitly or implicitly, in the WHEN phrase, all preceding ASCENDING/DESCENDING KEY data-names for identifier-1 must also be specified.

## SEARCH Statement

### Search Statement Considerations

Index data items cannot be used as subscripts, because of the restrictions on direct reference to them.

When the object of the VARYING option is an index-name for another table element, one Format 1 SEARCH statement steps through two table elements at once.

To ensure correct execution of a SEARCH statement for a variable-length table, make sure the object of the OCCURS DEPENDING ON clause (data-name-1) contains a value that specifies the current length of the table.

The scope of a SEARCH statement can be terminated by any of the following:

- An END-SEARCH phrase at the same level of nesting
- A separator period
- An ELSE or END-IF phrase associated with a previous IF statement

## SET Statement

The SET statement is used to perform one of the following operations:

- Placing values associated with table elements into indexes associated with index-names
- Incrementing or decrementing an occurrence number
- Setting the status of an external switch to ON or OFF
- Moving data to condition names to make conditions true
- Setting USAGE IS POINTER data items to a data address
- Setting USAGE IS PROCEDURE-POINTER data items to an entry address
- Setting USAGE OBJECT REFERENCE data items to refer to an object instance

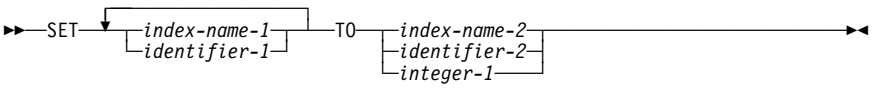
Index-names are related to a given table through the INDEXED BY phrase of the OCCURS clause; they are not further defined in the program.

When the sending and receiving fields in a SET statement share part of their storage (that is, the operands overlap), the result of the execution of such a SET statement is undefined.

## Format 1: SET for Basic Table Handling

When this form of the SET statement is executed, the current value of the receiving field is replaced by the value of the sending field (with conversion).

**Format 1—SET (Basic Table Handling)**




**index-name-1, identifier-1**
Receiving fields.

Must name either index data items or elementary numeric integer items.

**index-name-2**
Sending field.

The value before the SET statement is executed must correspond to the occurrence number of its associated table.

**identifier-2**
Sending field.

Must name either an index data item or an elementary numeric integer item.

**integer-1**
Sending field.

Must be a positive integer.

Table 43 shows valid combinations of sending and receiving fields in a Format 1 SET statement.

*Table 43. Sending and Receiving Fields for Format 1 SET Statement*

| | Receiving Field | | |
|---|---|---|---|
| **Sending Field** | **Index-name** | **Index Data Item** | **Integer Data Item** |
| Index-name | Valid | Valid* | Valid |
| Index Data Item | Valid* | Valid* | — |
| Integer Data Item | Valid | — | — |
| Integer Literal | Valid | — | — |

*No conversion takes place

Receiving fields are acted upon in the left-to-right order in which they are specified. Any subscripting or indexing associated with an identifier's receiving field is evaluated immediately before the field is acted upon.

The value used for the sending field is the value at the beginning of SET statement execution.

The value for an index-name after execution of a SEARCH or PERFORM statement can be undefined; therefore, a Format 1 SET statement should reinitialize such index-names before other table-handling operations are attempted.

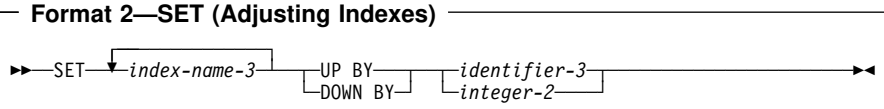
x If index-name-2 is for a table that has a subordinate item that contains an OCCURS
x DEPENDING ON clause, then undefined values can be received into identifier-1.

x For more information on complex OCCURS DEPENDING ON, see the *IBM COBOL*
x *Programming Guide* for your platform.

## Format 2: SET for Adjusting Indexes

When this form of the SET statement is executed, the value of the receiving field is increased (UP BY) or decreased (DOWN BY) by a value that corresponds to the value in the sending field.

**Format 2—SET (Adjusting Indexes)**




The **receiving field** can be specified by index-name-3. This index-name value both before and after the SET statement execution must correspond to the occurrence numbers in an associated table.

The **sending field** can be specified as identifier-3, which must be an elementary integer data item, or as integer-2, which must be a nonzero integer.

When the Format 2 SET statement is executed, the contents of the receiving field are increased (UP BY) or decreased (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of identifier-3 or integer-2. Receiving fields are acted upon in the left-to-right order in which they are specified. The value of the incrementing or decrementing field at the beginning of SET statement execution is used for all receiving fields.

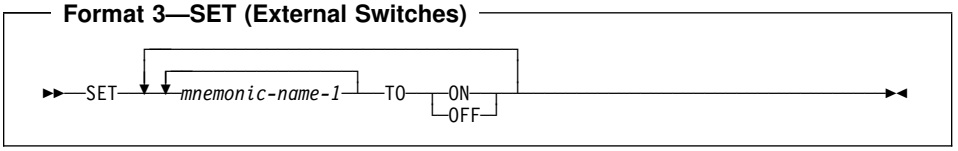
x If index-name-3 is for a table that has a subordinate item that contains an OCCURS
x DEPENDING ON clause, and if the ODO object is changed before executing a Format
x 2 SET Statement, then index-name-3 cannot contain a value that corresponds to an
x occurrence number of its associated table.

x For more information on complex OCCURS DEPENDING ON, see the *IBM COBOL*
x *Programming Guide* for your platform.

## Format 3: SET for External Switches

When this form of the SET statement is executed, the status of each external switch associated with the specified mnemonic-name is turned ON or OFF.

**Format 3—SET (External Switches)**

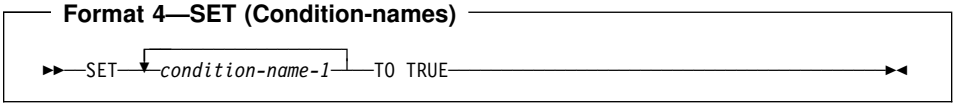



**mnemonic-name**
Must be associated with an external switch, the status of which can be altered.

## Format 4: SET for Condition-names

When this form of the SET statement is executed, the value associated with a condition-name is placed in its conditional variable according to the rules of the VALUE clause.

**Format 4—SET (Condition-names)**




**condition-name-1**
Must be associated with a conditional variable.

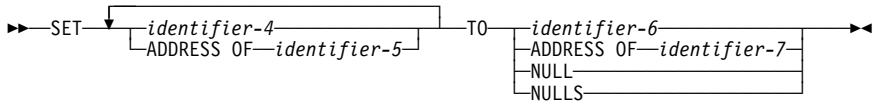
If more than one literal is specified in the VALUE clause of condition-name-1, its associated conditional variable is set equal to the first literal.

If multiple condition-names are specified, the results are the same as if a separate SET statement had been written for each condition-name in the same order in which they are specified in the SET statement.

## Format 5: SET for USAGE IS POINTER Data Items

When this form of the SET statement is executed, the current value of the receiving field is replaced by the address value contained in the sending field.

**Format 5—SET (USAGE IS POINTER Data Items)**




**identifier-4**
Receiving fields.

Must be described as USAGE IS POINTER.

**ADDRESS OF identifier-5**
Receiving fields.

**identifier-5** must be level-01 or level-77 items defined in the Linkage Section. The addresses of these items are set to the value of the operand specified in the TO phrase.

Identifier-5 must not be reference-modified.

**identifier-6**
Sending field.

Must be described as USAGE IS POINTER.

Cannot contain an address within the program's own Working-Storage, File, or Local-Storage Section.

**ADDRESS OF identifier-7**
Sending field.

Under AIX, OS/2, and Windows, it must name an item in either the Linkage Section or the Working-Storage Section of level 01, 77, or 02-49.

For MVS and VM, it must name an item in the Linkage Section of any level except 66 or 88. **ADDRESS OF identifier-7** contains the address of the identifier, and not the content of the identifier.

**NULL**
**NULLS**
Sending field.

Sets the receiving field to contain the value of an invalid address.

Table 44 shows valid combinations of sending and receiving fields in a Format 5 SET statement.

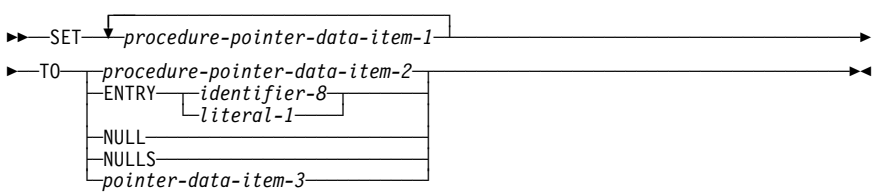
*Table 44. Sending and Receiving Fields for Format 5 SET Statement*

| Sending Field | Receiving Field | | |
|---|---|---|---|
| | USAGE IS POINTER | ADDRESS OF | NULL/NULLS |
| USAGE IS POINTER | Valid | Valid | - |
| ADDRESS OF | Valid | Valid | - |
| NULL/NULLS | Valid | Valid | - |

## Format 6: SET for USAGE IS PROCEDURE-POINTER Data Items

When this format of the SET statement is executed, the current value of the receiving field is replaced by the address value contained in the sending field. Additionally, to enable COBOL programs to interoperate with C programs via C function pointers, the sending field can be a pointer. The pointer is converted to a procedure-pointer and is stored in the receiver.

**Format 6—SET (USAGE IS PROCEDURE-POINTER Data Items)**




**procedure-pointer-data-item-1, procedure-pointer-data-item-2**
Must be described as USAGE IS PROCEDURE-POINTER.
Procedure-pointer-data-item-1 is the receiving field.

**identifier-8**
Must be defined as an alphanumeric item such that the value can be a program name. For more information, see "PROGRAM-ID Paragraph" on page 75. For entry points in non-COBOL programs, identifier-8 can contain the characters @, #, and, $.

**literal-1**
Must be nonnumeric and must conform to the rules for formation of program-names. For details on formation rules, see the discussion of program-name under "PROGRAM-ID Paragraph" on page 75.

Identifier-8 or literal-1 must refer to one of the following types of entry points:

- The primary entry point of a COBOL program as defined by the PROGRAM-ID statement. The PROGRAM-ID must reference the outermost program of a compilation unit; it must not reference a nested program.
- An alternate entry point of a COBOL program as defined by a COBOL ENTRY statement.
- An entry point in a non-COBOL program.

The program-name referenced by the SET...TO ENTRY statement can be affected by the PGMNAME compiler option. For details, see the *IBM COBOL Programming Guide* for your platform.

**NULL**
**NULLS**
Sets the receiving field to contain the value of an invalid address.

**pointer-data-item-3**
Must be defined with USAGE POINTER. You must set pointer-data-item-3 in a non-COBOL program, to point to a valid program entry point.

### Example of COBOL/C Interoperability

The following example demonstrates a COBOL CALL to a C function that returns a function-pointer to a service, followed by a COBOL CALL to the service:

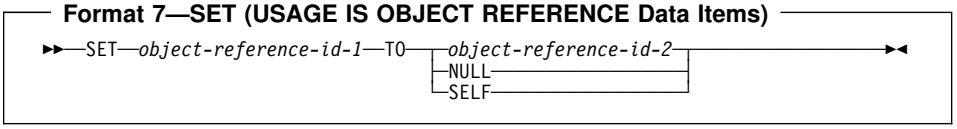
```
IDENTIFICATION DIVISION.
PROGRAM-ID DEMO.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 FP USAGE POINTER.
01 PP USAGE PROCEDURE-POINTER.
PROCEDURE DIVISION.
    CALL "c-function" RETURNING FP.
    SET PP TO FP.
    CALL PP.
```

For MVS and VM, COBOL PROCEDURE-POINTERs are 8-bytes in length. Thus, the SET statement is needed to convert the function pointer (FP) to the COBOL PROCEDURE-POINTER (PP).

## Format 7: SET for USAGE OBJECT REFERENCE Data Items

When this format of the SET statement is executed the value in the receiving item is replaced by the value in the sending item.

**Format 7—SET (USAGE IS OBJECT REFERENCE Data Items)**




Object-reference-id-1 and object-reference-id-2 must be defined as USAGE OBJECT REFERENCE, with object-reference-id-1 being the receiver and object-reference-id-2 being the sender. If object-reference-id-1 is defined as an object reference of a certain class (defined as "USAGE OBJECT REFERENCE class-name"), object-reference-id-2 must be an object reference of the same class or a class derived from that class.

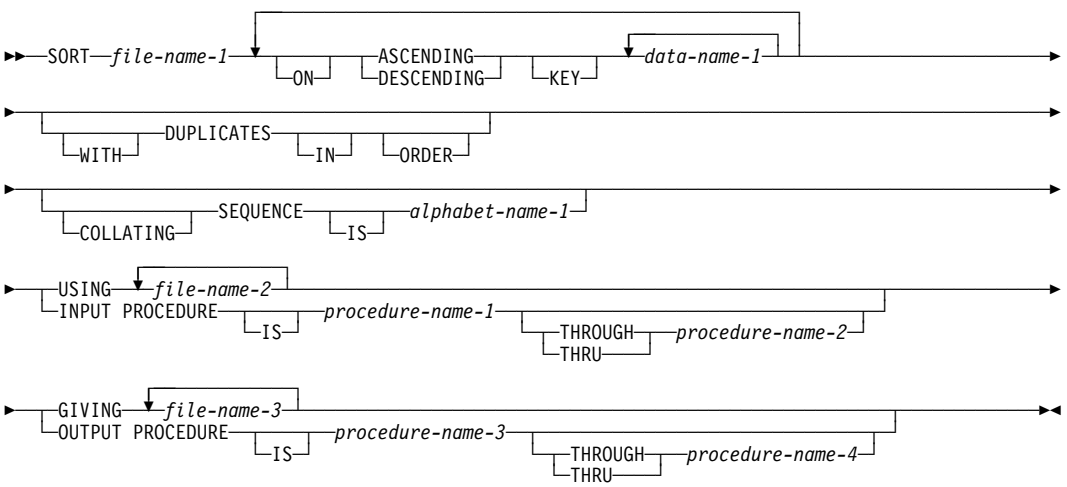
If the figurative constant NULL is specified, the receiving object-reference-id-1 is set to the NULL value.

If SELF is specified, the SET statement must appear in the procedure division of a method. In this case, object-reference-id-1 is set to refer to the object upon which the currently executing method was invoked.

## SORT Statement

The SORT statement accepts records from one or more files, sorts them according to the specified key(s), and makes the sorted records available either through an OUTPUT PROCEDURE or in an output file. See also “MERGE Statement” on page 327. The SORT statement can appear anywhere in the Procedure Division except in the declarative portion.

**Format**




**file-name-1**

The name given in the SD entry that describes the records to be sorted.

No pair of file-names in a SORT statement can be specified in the same SAME SORT AREA, or SAME SORT-MERGE AREA clause. File-names associated with the GIVING clause (file-name-3...) cannot be specified in the SAME AREA clause.

x File-names associated with the GIVING clause (file-name-3...) can be specified in the
x SAME AREA clause.

## ASCENDING/DESCENDING KEY Phrase

This phrase specifies that records are to be processed in ascending or descending sequence (depending on the phrase specified), based on the specified sort keys.

**data-name-1**

Specifies a KEY data item on which the SORT statement will be based. Each such data-name must identify a data item in a record associated with **file-name-1**. The data-names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY phrases. The left-most data-name is the major key, the next data-name is the next most significant key, and so forth. The following rules apply:

1. A specific KEY data item must be physically located in the same position and have the same data format in each input file. However, it need not have the same data-name.
2. If file-name-1 has more than one record description, then the KEY data items need be described in only one of the record descriptions.
3. If file-name-1 contains variable-length records, all of the KEY data-items must be contained within the first *n* character positions of the record, where *n* equals the minimum records size specified for file-name-1.
4. KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.
5. KEY data items can be qualified.
6. KEY data items cannot be group items that contain variable occurrence data items.
7. KEY data items can be floating-point items.
8. KEY data items cannot be variably-located.

If file-name-3 references an indexed file, the first specification of data-name-1 must be associated with an ASCENDING phrase and the data item referenced by that data-name-1 must occupy the same character positions in this record as the data item associated with the major record key for that file.

The direction of the sorting operation depends on the specification of the ASCENDING or DESCENDING key words as follows:

- When ASCENDING is specified, the sequence is from the lowest key value to the highest key value.
- When DESCENDING is specified, the sequence is from the highest key value to the lowest.
- If the KEY data item is alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited, the sequence of key values depends on the collating sequence used (see “COLLATING SEQUENCE Phrase” on page 390).

- MVS/VM Under MVS and VM, if the KEY is a DBCS item, the sequence of the KEY values are based on the binary collating sequence of the hexadecimal values of the DBCS characters.
- WRKSTN Under AIX, OS/2, and Windows, if the KEY is a DBCS item, the sequence of the KEY values are based on the collating sequence as specified by the locale. For more information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 543.
- If the KEY is an external floating-point item, the compiler will treat the data item as character data, rather than numeric data. The sequence in which the records are sorted depends on the collating sequence used.
- If the KEY data item is internal floating-point, the sequence of key values will be in numeric order.
- The key comparisons are performed according to the rules for comparison of operands in a relation condition (see "Relation Condition" under "Conditional Expressions" on page 225).

## DUPLICATES Phrase

If the DUPLICATES phrase is specified, and the contents of all the key elements associated with one record are equal to the corresponding key elements in one or more other records, the order of return of these records is as follows:

- The order of the associated input files as specified in the SORT statement. Within a given file the order is that in which the records are accessed from that file.
- The order in which these records are released by an input procedure, when an input procedure is specified.

If the DUPLICATES phrase is not specified, the order of these records is undefined. For more information about use of the DUPLICATES phrase, see the related discussion of alternate indexes in the *IBM COBOL Programming Guide* for your platform.

## COLLATING SEQUENCE Phrase

This phrase specifies the collating sequence to be used in nonnumeric comparisons for the KEY data items in this sorting operation.

**alphabet-name-1**

Must be specified in the ALPHABET clause of the SPECIAL-NAMES paragraph. Any one of the alphabet-name clause phrases can be specified with the following results:

**STANDARD-1**

MVS/VM Under MVS and VM, the ASCII collating sequence is used for all nonnumeric comparisons. (The ASCII collating sequence is in Appendix B, "EBCDIC and ASCII Collating Sequences" on page 522.)

WRKSTN Under AIX, OS/2, and Windows the collating sequence is based on the character's hex value order.

**STANDARD-2**

MVS/VM Under MVS and VM, the International Reference Version of the ISO 7-bit code defined in International Standard 646, 7-bit Coded Character Set for Information Processing Interchange is used for all nonnumeric comparisons.

WRKSTN Under AIX, OS/2, and Windows, the collating sequence is based on the character's hex value order.

**NATIVE**

MVS/VM Under MVS and VM, the EBCDIC collating sequence is used for all nonnumeric comparisons. (The EBCDIC collating sequence is in Appendix B, "EBCDIC and ASCII Collating Sequences" on page 522.)

WRKSTN Under AIX, OS/2, and Windows, the collating sequence indicated by the locale is selected.

**EBCDIC**

The EBCDIC collating sequence is used for all nonnumeric comparisons. (The EBCDIC collating sequence is in Appendix B, "EBCDIC and ASCII Collating Sequences" on page 522.)

**literal**

The collating sequence established by the specification of literals in the alphabet-name clause is used for all nonnumeric comparisons.

When the COLLATING SEQUENCE phrase is omitted, the PROGRAM COLLATING SEQUENCE clause (if specified) in the OBJECT-COMPUTER paragraph specifies the collating sequence to be used. When both the COLLATING SEQUENCE phrase and the PROGRAM COLLATING SEQUENCE clauses are omitted, the EBCDIC collating sequence is used.

## USING Phrase

**file-name-2,...**

The input files.

When the USING phrase is specified, all the records in file-name-2,..., (that is, the input files) are transferred automatically to file-name-1. At the time the SORT statement is executed, these files must not be open; the compiler opens, reads, makes records available, and closes these files automatically. If EXCEPTION/ERROR procedures are specified for these files, the compiler makes the necessary linkage to these procedures.

All input files must be described in FD entries in the Data Division.

If the USING phrase is specified and if file-name-1 contains variable-length records, the size of the records contained in the input files (file-name-2,...) must not be less than the smallest record nor greater than the largest record described for file-name-1. If file-name-1 contains fixed-length records, the size of the records contained in the input files must not be greater than the largest record described for file-name-1. For more information, see the *IBM COBOL Programming Guide* for your platform.

## SORT Statement

## INPUT PROCEDURE Phrase

This phrase specifies the name of a procedure that is to select or modify input records before the sorting operation begins.

**procedure-name-1**
Specifies the first (or only) section or paragraph in the INPUT PROCEDURE.

**procedure-name-2**
Identifies the last section or paragraph of the INPUT PROCEDURE.

The input procedure can consist of any procedure needed to select, modify, or copy the records that are made available one at a time by the RELEASE statement to the file referenced by file-name-1. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements 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. The range of the input procedure must not cause the execution of any MERGE, RETURN, or SORT statement.

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 at the end of the last statement in the input procedure. When control passes the last statement in the input procedure, the records that have been released to the file referenced by file-name-1 are sorted.

## GIVING Phrase

**file-name-3,...**
The output files.

When the GIVING phrase is specified, all the sorted records in file-name-1 are automatically transferred to the output files (file-name-3,...).

All output files must be described in FD entries in the Data Division.

If the output files (file-name-3,...) contain variable-length records, the size of the records contained in file-name-1 must not be less than the smallest record nor greater than the largest record described for the output files. If the output files contain fixed-length records, the size of the records contained in file-name-1 must not be greater than the largest record described for the output files. For more information, see the *IBM COBOL Programming Guide* for your platform.

At the time the SORT statement is executed, the output files (file-name-3,...) must not be open. For each of the output files, the execution of the SORT statement causes the following actions to be taken:

- The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed.
- 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.

For a relative file, the relative key data item for the first record returned contains 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.

- 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 must 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 STANDARD EXCEPTION/ERROR procedure specified for the file 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.

## OUTPUT PROCEDURE Phrase

This phrase specifies the name of a procedure that is to select or modify output records from the sorting operation.

**procedure-name-3**
Specifies the first (or only) section or paragraph in the OUTPUT PROCEDURE.

**procedure-name-4**
Identifies the last section or paragraph of the OUTPUT PROCEDURE.

The output procedure can 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 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 by CALL, EXIT, GO TO, and PERFORM statements in the range of the output procedure. The range also includes 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 must not cause the execution of any MERGE, RELEASE, or SORT statement.

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 at the end of the last statement in the output procedure and when control passes the last statement in the output procedure, the return mechanism provides 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 can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

## SORT Statement

**Note:** The INPUT and OUTPUT PROCEDURE phrases are similar to those for a basic PERFORM statement. For example, if you name a procedure in an OUTPUT PROCEDURE, that procedure is executed during the sorting operation just as if it were named in a PERFORM statement. As with the PERFORM statement, execution of the procedure is terminated after the last statement completes execution. The last statement in an INPUT or OUTPUT PROCEDURE can be the EXIT statement (see "EXIT Statement" on page 299).

## SORT Special Registers

The special registers, SORT-CORE-SIZE, SORT-MESSAGE, and SORT-MODE-SIZE, are equivalent to option control statement key words in the sort control file. You define the sort control data set with the SORT-CONTROL special register.

**Note:** If you use a sort control file to specify control statements, the values specified in the sort control file take precedence over those in the special register.

**SORT-MESSAGE Special Register**
See "SORT-MESSAGE" on page 20.

**SORT-CORE-SIZE Special Register**
See "SORT-CORE-SIZE" on page 19.

**SORT-FILE-SIZE Special Register**
See "SORT-FILE-SIZE" on page 20.

**SORT-MODE-SIZE Special Register**
See "SORT-MODE-SIZE" on page 21.

**SORT-CONTROL Special Register**
See "SORT-CONTROL" on page 18.

**SORT-RETURN Special Register**
See "SORT-RETURN" on page 21.

## Segmentation Considerations

If the SORT statement appears in a section that is not in an independent segment, then any input or output procedure referenced by that SORT statement must appear:

- Totally within non-independent segments, or
- Wholly contained in a single independent segment.

If a SORT statement appears in an independent segment, then any input or output procedure referenced by that SORT statement must be contained:

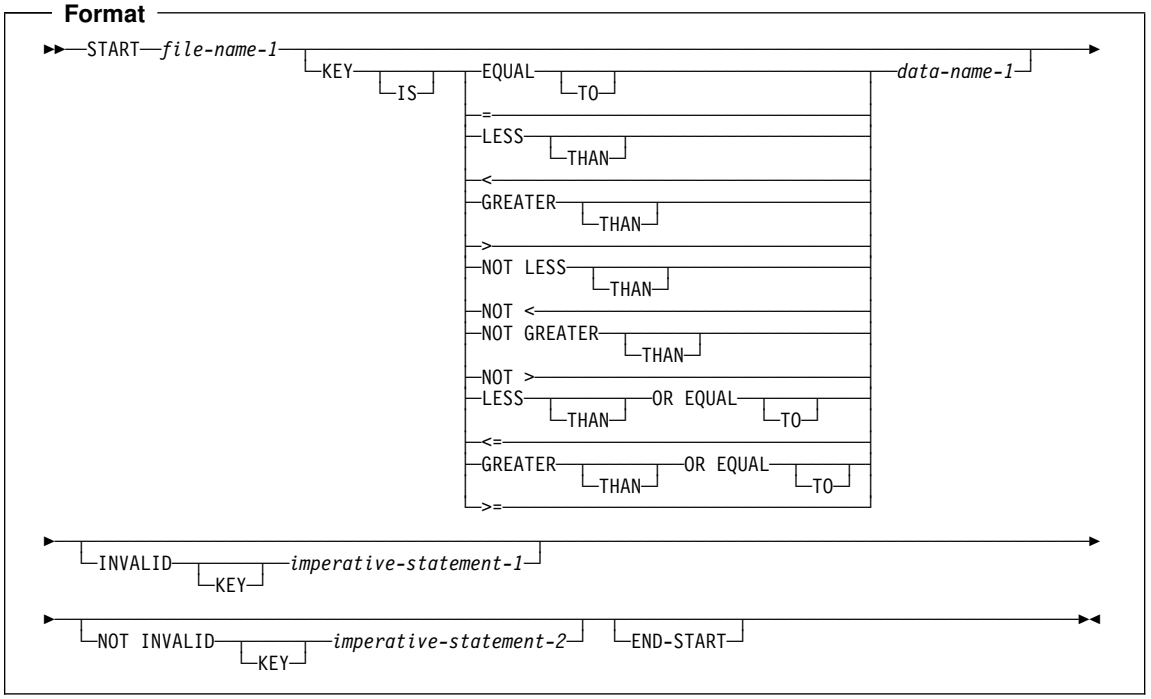
- Totally within non-independent segments, or
- Wholly within the same independent segment as that SORT statement.

## START Statement

The START statement provides a means of positioning within an indexed or relative file for subsequent sequential record retrieval.

When the START statement is executed, the associated indexed or relative file must be open in either INPUT or I-O mode.

**Format**




**file-name-1**

Must name a file with sequential or dynamic access. File-name-1 must be defined in an FD entry in the Data Division, and must not name a sort file.

## KEY Phrase

When the KEY phrase is specified, the file position indicator is positioned at the logical record in the file whose key field satisfies the comparison.

When the KEY phrase is not specified, KEY IS EQUAL (to the prime record key) is implied.

x WRKSTN Under AIX, OS/2, and Windows, the following relational operators are
x allowed in the KEY phase:

| | | |
|---|---|---|
| x | LESS THAN | < |
| x | NOT GREATER THAN | NOT > |
| x | LESS THAN OR EQUAL TO | <= |

If you specify the KEY to be 'less than', or 'less than or equal to' the data item, the file position indicator is positioned to the last logical record currently existing in the file satisfying the comparison.

For an indexed file, if the key that satisfies the comparison has duplicate entries, the file position indicator is positioned to the last of these entries.

**data-name-1**
May be qualified; it may not be subscripted.

When the START statement is executed, a comparison is made between the current value in the key data-name and the corresponding key field in the file's index.

If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the START statement is executed (See “Status Key” on page 254).

## INVALID KEY Phrases

If the comparison is not satisfied by any record in the file, an invalid key condition exists; the position of the file position indicator is undefined, and (if specified) the INVALID KEY imperative-statement is executed. (See “Invalid Key Condition” under “Common Processing Facilities” on page 254.)

The INVALID KEY phrase must be specified if no EXCEPTION/ERROR procedure is explicitly or implicitly specified for this file .

Both the INVALID KEY phrase and the EXCEPTION/ERROR procedure may be omitted.

## END-START Phrase

This explicit scope terminator serves to delimit the scope of the START statement. END-START permits a conditional START statement to be nested in another conditional statement. END-START may also be used with an imperative START statement.

For more information, see “Delimited Scope Statements” on page 247.

## Effect of Record Locking (LOCK MODE Clause)

WRKSTN Under AIX, OS/2, and Windows, the START statement cannot acquire or detect a record lock. However, if the lock mode is AUTOMATIC, a successful START statement releases an existing record lock. ◁□

## Indexed Files

When the KEY phrase is specified, the key data item used for the comparison is data-name.

When the KEY phrase is not specified, the key data item used for the EQUAL TO comparison is the prime RECORD KEY.

When START statement execution is successful, the RECORD KEY or ALTERNATE RECORD KEY with which data-name is associated becomes the key of reference for subsequent READ statements.

**data-name-1**
Can be any of the following:

- The prime RECORD KEY
- Any ALTERNATE RECORD KEY
- An alphanumeric data item within a record description for a file whose leftmost character position corresponds to the leftmost character position of that record key; it may be qualified. The data item must be less than or equal to the length of the record key for the file.

  x Need not be an alphanumeric item. However, for purposes of the I/O operax tion, it will be treated as an alphanumeric item.

The file position indicator points to the first record in the file whose key field satisfies the comparison. If the operands in the comparison are of unequal lengths, the comparison proceeds as if the longer field were truncated on the right to the length of the shorter field. All other numeric and nonnumeric comparison rules apply, except that the PROGRAM COLLATING SEQUENCE clause, if specified, has no effect.

When START statement execution is successful, the RECORD KEY with which data-name-1 is associated becomes the key of reference for subsequent READ statements.

When START statement execution is unsuccessful, the key of reference is undefined.

## Relative Files

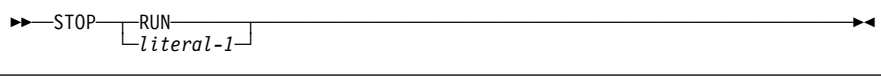
When the KEY phrase is specified, data-name-1 must specify the RELATIVE KEY.

Whether or not the KEY phrase is specified, the key data item used in the comparison is the RELATIVE KEY data item. The file position indicator points to the logical record in the file whose key satisfies the comparison.

## STOP Statement

The STOP statement halts execution of the object program either permanently or temporarily.

**Format**




**literal**

May be numeric or nonnumeric, and may be any figurative constant except ALL literal. If the literal is numeric, it must be an unsigned integer.

x May be a signed numeric integer or non-integer literal, but may not be a floating-
x point literal.

When STOP literal is specified, the literal is communicated to the operator, and object program execution is suspended. Program execution is resumed only after operator intervention, and continues at the next executable statement in sequence.

The STOP literal statement is useful for special situations (a special tape or disk must be mounted, a specific daily code must be entered, and so forth) when operator intervention is needed during program execution. However, the ACCEPT and DISPLAY statements are preferred when operator intervention is needed.

x WRKSTN Under AIX, OS/2, and Windows, do not use the STOP RUN or STOP literal
x statement in programs compiled with the THREAD compiler option.

When STOP RUN is specified, execution of the object program is terminated, and control is returned to the system. If a STOP RUN statement appears in a sequence of imperative statements within a sentence, it must be the last or only statement in the sequence.

x The STOP RUN statement does not have to be the last statement in a sequence, but
x the statements following the STOP RUN will not be executed.

## STOP Statement

The STOP RUN statement closes **all** files defined in any of the programs comprising the run unit.

For use of the STOP RUN statement in calling and called programs, see the table below.

| Termination Statement | Main Program | Subprogram |
|---|---|---|
| STOP RUN | Return to calling program.* (May be the system and cause the job to end.) | Return directly to the program that called the main program.* (May be the system and cause the job to end.) |

MVS/VM Under MVS and VM, if the main program is called by a program written in a language that does not follow COBOL linkage conventions, return will be to this calling program. See *Language Environment Programming Guide* for system specific behavior.

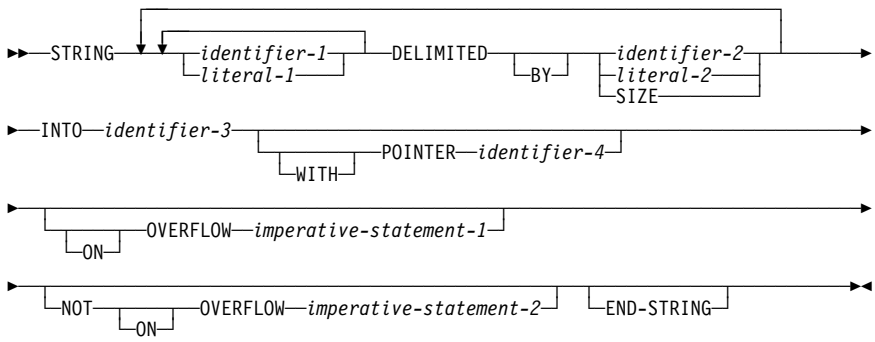
**Note:** When running on MVS/ESA and using the ESA/370 linkage stack instructions, STOP RUN cannot be used to terminate a COBOL run unit from a subprogram if the ESA/370 linkage stack level has changed from the time the COBOL main program for that run unit was entered.

## STRING Statement

The STRING statement strings together the partial or complete contents of two or more data items or literals into one single data item.

One STRING statement can be written instead of a series of MOVE statements.

**Format**




**identifier-1**
Represents the **sending field(s)**.

All identifiers (except identifier-4, the POINTER item) must have USAGE DISPLAY, explicitly or implicitly.

When the sending field or any of the delimiters is an elementary numeric item, it must be described as an integer, and its PICTURE character-string must not contain the symbol P.

**literal-1**
Represents the **sending field(s)**.

All literals must be nonnumeric literals; each may be any figurative constant except the ALL literal. When a figurative constant is specified, it is considered a 1-character nonnumeric literal.

**identifier-1 through identifier-3**
x May be DBCS data items. If one of these identifiers is a DBCS item, then all of
x them, and all literals, must be DBCS items. May not be external floating-point
x items.

x **literal-1 and literal-2**
x May be DBCS literals. If one of these is a DBCS literal, then all of them must be
x DBCS literals, and identifier-1 through identifier-3 must be DBCS items.

x SPACE is the only figurative constant allowed for DBCS items.

## DELIMITED BY Phrase

The DELIMITED BY phrase sets the limits of the string.

**identifier-2, literal-2**
Are delimiters; that is, character(s) that delimit the data to be transferred.

If identifier-1 or identifier-2 occupies the same storage area as identifier-3 or identifier-4, undefined results will occur, even if the identifiers are defined by the same data description entry.

When a figurative constant is specified, it is considered a 1-character nonnumeric literal.

**SIZE**
Transfers the complete sending area.

## INTO Phrase

**identifier-3**
Represents the **receiving field**.

x It must not represent an edited data item or external floating-point item and must not be described with the JUSTIFIED clause. It must not be reference-modified.

If identifier-3 and identifier-4 occupy the same storage area, undefined results will occur, even if the identifiers are defined by the same data description entry.

## POINTER Phrase

**identifier-4**
Represents the **pointer field**, which points to a character position in the receiving field.

It must be an elementary integer data item large enough to contain a value equal to the length of the receiving area plus 1. The pointer field must not contain the symbol P in its PICTURE character-string.

x When identifier-3 (the receiving field) is a DBCS data item, identifier-4 indicates the
x relative DBCS character position (not the relative byte position) in the receiving
x field.

## ON OVERFLOW Phrases

**imperative-statement-1**
Executed when the pointer value (explicit or implicit):

- Is less than 1
- Exceeds a value equal to the length of the receiving field.

When either of the above conditions occurs, an overflow condition exists, and no more data is transferred. Then the STRING operation is terminated, the NOT ON OVERFLOW phrase, if specified, is ignored, and control is transferred to the end of the STRING statement or, if the ON OVERFLOW phrase is specified, to imperative-statement-1.

If control is transferred to imperative-statement-1, 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 STRING statement.

If at the time of execution of a STRING statement, conditions that would cause an overflow condition are not encountered, then after completion of the transfer of data, the ON OVERFLOW phrase, if specified, is ignored. Control is then 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 according to 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.

## END-STRING Phrase

This explicit scope terminator serves to delimit the scope of the STRING statement. END-STRING permits a conditional STRING statement to be nested in another conditional statement. END-STRING may also be used with an imperative STRING statement.

For more information, see “Delimited Scope Statements” on page 247.

## Data Flow

When the STRING statement is executed, data is transferred from the sending fields to the receiving field. The order in which sending fields are processed is the order in which they are specified. The following rules apply:

- Characters from the sending fields are transferred to the receiving field, according to the rules for alphanumeric to alphanumeric elementary moves, except that no space filling is provided (see “MOVE Statement” on page 333).
- When DELIMITED BY identifier/literal is specified, the contents of each sending item are transferred, character-by-character, beginning with the leftmost character and continuing until either:
  - A delimiter for this sending field is reached (the delimiter itself is not transferred), or
  - The rightmost character of this sending field has been transferred.
- When DELIMITED BY SIZE identifier is specified, each entire sending field is transferred to the receiving field.
- When the receiving field is filled, or when all the sending fields have been processed, the operation is ended.

- When the POINTER phrase is specified, an explicit pointer field is available to the COBOL user to control placement of data in the receiving field. The user must set the explicit pointer's initial value, which must not be less than 1 and not more than the character count of the receiving field. (Note that the pointer field must be defined as a field large enough to contain a value equal to the length of the receiving field plus 1; this precludes arithmetic overflow when the system updates the pointer at the end of the transfer.)
- When the POINTER phrase is not specified, no pointer is available to the user. However, a conceptual implicit pointer with an initial value of 1 is used by the system.
- Conceptually, when the STRING statement is executed, the initial pointer value (explicit or implicit) is the first character position within the receiving field into which data is to be transferred. Beginning at that position, data is then positioned, character-by-character, from left to right. After each character is positioned, the explicit or implicit pointer is increased by 1. The value in the pointer field is changed only in this manner. At the end of processing, the pointer value always indicates a value equal to one character beyond the last character transferred into the receiving field.

**Note:** Subscript, reference modification, variable-length or variable location calculations, and function evaluations are performed only once, at the beginning of the execution of the STRING statement. Therefore, if identifier-3 or identifier-4 is used as a subscript, reference-modifier, or function argument in the STRING statement, or affects the length or location of any of the identifiers in the STRING statement, these values are determined at the beginning of the STRING statement, and are **not** affected by any results of the STRING statement.

After STRING statement execution is completed, only that part of the receiving field into which data was transferred is changed. The rest of the receiving field contains the data that was present before this execution of the STRING statement.

When the following STRING statement is executed, the results obtained will be like those illustrated in Figure 18 on page 404.

```
STRING ID-1 ID-2 DELIMITED BY ID-3
       ID-4 ID-5 DELIMITED BY SIZE
  INTO ID-7 WITH POINTER ID-8
END-STRING
```

## STRING Statement

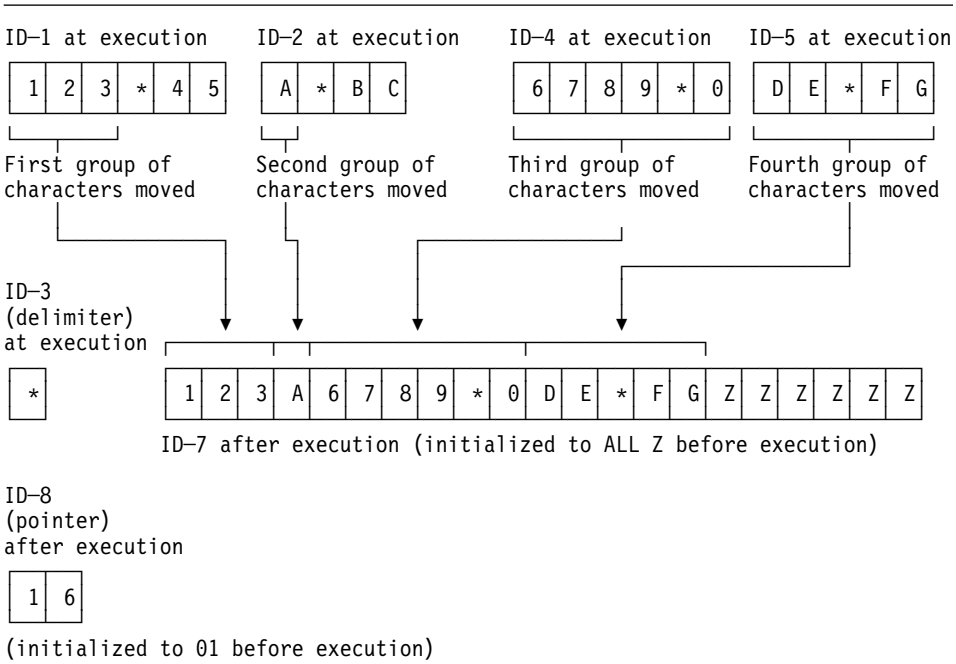
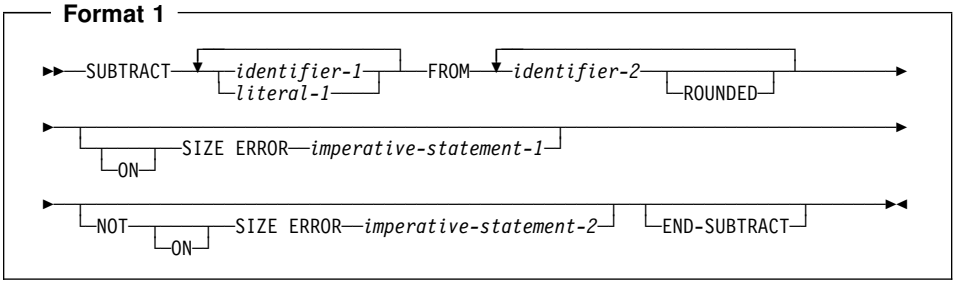



*Figure 18. STRING Statement Execution Results*

## SUBTRACT Statement

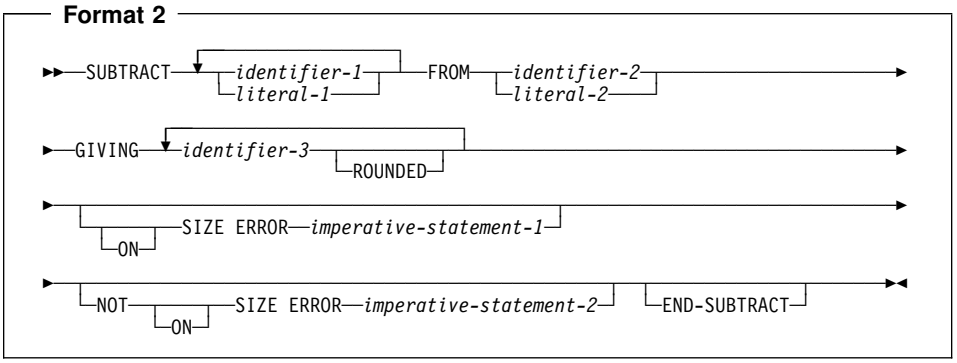
The SUBTRACT statement subtracts one numeric item, or the sum of two or more numeric items, from one or more numeric items, and stores the result.

**Format 1**



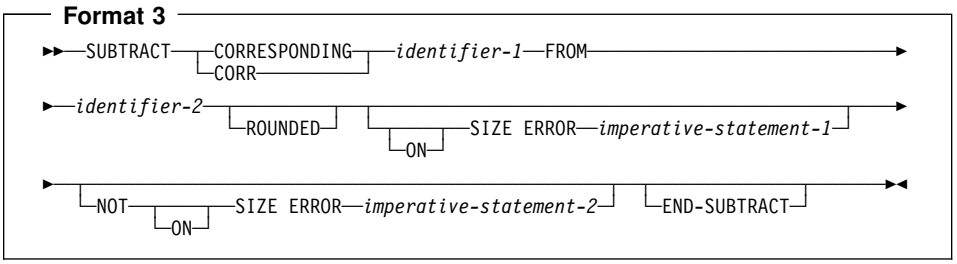

All identifiers or literals preceding the key word FROM are added together and this sum is subtracted from and stored immediately in identifier-2. This process is repeated for each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

**Format 2**




All identifiers or literals preceding the key word FROM are added together and this sum is subtracted from identifier-2 or literal-2. The result of the subtraction is stored as the new value of each data item referenced by identifier-3.

**Format 3**



## SUBTRACT Statement

Elementary data items within identifier-1 are subtracted from, and the results are stored in, the corresponding elementary data items within identifier-2.

The composite of operands must not contain more than 18 digits. The compiler ensures that enough places are carried so that no significant digits are lost during execution.

x The composite of operands may contain more than 18 digits. For more information on
x arithmetic intermediate results, see the *IBM COBOL Programming Guide* for your plat-
x form.

For all Formats:

**identifier**
In Format 1, must name an elementary numeric item.

In Format 2, must name an elementary numeric item, unless the identifier follows the word GIVING. Each identifier following the word GIVING must name a numeric or numeric-edited elementary item.

In Format 3, must name a group item.

**literal**
Must be a numeric literal.

x Floating-point data items and literals may be used anywhere numeric data items and
x literals can be specified.

## ROUNDED Phrase

For information on the ROUNDED phrase, and for operand considerations, see "ROUNDED Phrase" on page 250.

## SIZE ERROR Phrases

For information on the SIZE ERROR phrases, and for operand considerations, see "SIZE ERROR Phrases" on page 251.

## CORRESPONDING Phrase (Format 3)

See "CORRESPONDING Phrase" on page 249.

## END-SUBTRACT Phrase

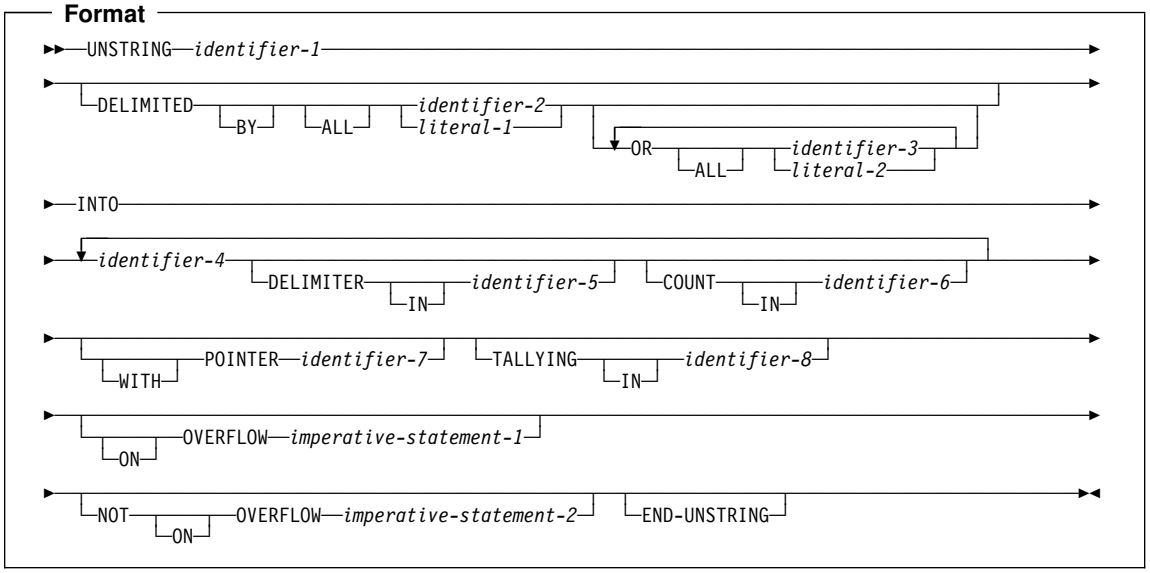
This explicit scope terminator serves to delimit the scope of the SUBTRACT statement. END-SUBTRACT permits a conditional SUBTRACT statement to be nested in another conditional statement. END-SUBTRACT may also be used with an imperative SUBTRACT statement.

For more information, see "Delimited Scope Statements" on page 247.

## UNSTRING Statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

**Format**




**identifier-1**

Represents the **sending field**. Data is transferred from this field to the data receiving fields (identifier-4).

It must be an alphanumeric data item; it cannot be reference-modified.

x It can be an alphanumeric-edited or an alphabetic data item. It can also be a
x DBCS data item.

One UNSTRING statement can take the place of a series of MOVE statements, except that evaluation or calculation of certain elements is performed only once, at the beginning of the execution of the UNSTRING statement. For more information, see “Values at the End of Execution of the UNSTRING Statement” on page 413.

The rules for moving an alphanumeric elementary item are the same as those for the MOVE statement (see “MOVE Statement” on page 333).

## DELIMITED BY Phrase

This phrase specifies delimiters within the data that control the data transfer.

If the DELIMITED BY phrase is *not* specified, the DELIMITER IN and COUNT IN phrases must *not* be specified.

**identifier-2**
**identifier-3**
Each represents one delimiter.

Each can be either of the following:

- An alphanumeric data item
- A DBCS data item

  If any are DBCS items, then all must be DBCS items. Figurative constants SPACE and SPACES are allowed for DBCS items.

**literal-1**
**literal-2**
Each represents one delimiter.

Each can either of the following:

- A nonnumeric literal
- A DBCS literal

  If any are DBCS literals, all must be DBCS literals. Figurative constants SPACE and SPACES are allowed for DBCS literals.

Each can be any figurative constant except the ALL literal. When a figurative constant is specified, it is considered to be a 1-character nonnumeric literal.

**ALL**
One or more contiguous occurrences of any delimiters are treated as if they were only one occurrence; this one occurrence is moved to the delimiter receiving field (identifier-5), if specified. The delimiting characters in the sending field are treated as an elementary alphanumeric or DBCS item and are moved into the current delimiter receiving field, according to the rules of the MOVE statement.

When DELIMITED BY ALL is *not* specified, and two or more contiguous occurrences of any delimiter are encountered, the current data receiving field (identifier-4) is filled with spaces or zeros, according to the description of the data receiving field.

## Delimiter with Two or More Characters

A delimiter that contains two or more characters is recognized as a delimiter only if the delimiting characters are both of the following:

- Contiguous
- In the sequence specified in the sending field

## Two or More Delimiters

When two or more delimiters are specified, an OR condition exists, and each nonoverlapping occurrence of any one of the delimiters is recognized in the sending field in the sequence specified.

For example:

```
DELIMITED BY "AB" or "BC"
```

An occurrence of either AB or BC in the sending field is considered a delimiter. An occurrence of ABC is considered an occurrence of AB.

## INTO Phrase

This phrase specifies the fields where the data is to be moved.

**identifier-4**
Represents the **data receiving fields**.

Each must have USAGE DISPLAY. These fields can be defined as any of the following:

- Alphabetic
- Alphanumeric
- Numeric (without the symbol P in the PICTURE string)—must *not* be defined as an alphanumeric-edited item or a numeric-edited item.
- DBCS

Identifier-4 cannot be defined as a floating-point item.

**DELIMITER IN**
If the DELIMITED BY phrase is *not* specified, the DELIMITER IN phrase must *not* be specified.

**identifier-5**
Represents the **delimiter receiving fields**. It can be:

- Alphanumeric
- A DBCS data item

**COUNT IN**
If the DELIMITED BY phrase is *not* specified, the COUNT IN phrase must *not* be specified.

**identifier-6**
Is the **data-count field** for each data transfer. Each field holds the count of examined characters in the sending field, terminated by the delimiters or the end of the sending field, for the move to this receiving field; the delimiters are not included in this count.

Identifier-6 must be an integer data item defined without the symbol P in the PICTURE string.

When identifier-1 (the sending field) is a DBCS data item, identifier-6 indicates the number of DBCS characters (not the number of bytes) examined in the sending field.

## POINTER Phrase

When the POINTER phrase is specified, the value of the pointer field behaves as if it were increased by 1 for each examined character in the sending field. When execution of the UNSTRING statement is completed, the pointer field contains a value equal to its initial value, plus the number of characters examined in the sending field.

When this phrase is specified, the user must initialize identifier-7 before execution of the UNSTRING statement begins.

**identifier-7**

Is the **pointer field**. This field contains a value that indicates a relative position in the sending field.

Identifier-7 must be an integer data item defined without the symbol P in the PICTURE string.

It must be described as a data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by identifier-1.

x When identifier-1 (the sending field) is a DBCS data item, identifier-7 indicates the
x relative DBCS character position (not the relative byte position) in the sending field.

## TALLYING IN Phrase

When the TALLYING phrase is specified, the field-count field contains (at the end of execution of the UNSTRING statement) a value equal to the initial value, plus the number of data receiving areas acted upon.

When this phrase is specified, the user must initialize identifier-8 before execution of the UNSTRING statement begins.

**identifier-8**

Is the **field-count field**. This field is increased by the number of data receiving fields acted upon in this execution of the UNSTRING statement.

It must be an integer data item defined without the symbol P in the PICTURE string.

## ON OVERFLOW Phrases

An overflow condition exists when:

- The pointer value (explicit or implicit) is less than 1.
- The pointer value (explicit or implicit) exceeds a value equal to the length of the sending field.
- All data receiving fields have been acted upon, and the sending field still contains unexamined characters.

### When an Overflow Condition Occurs

An overflow condition results in the following:

1. No more data is transferred.
2. The UNSTRING operation is terminated.
3. The NOT ON OVERFLOW phrase, if specified, is ignored.
4. Control is transferred to the end of the UNSTRING statement or, if the ON OVERFLOW phrase is specified, to imperative-statement-1.

**imperative-statement-1**

Statement or statements for dealing with an overflow condition.

If control is transferred to imperative-statement-1, 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 UNSTRING statement.

### When an Overflow Condition Does Not Occur

When, during execution of an UNSTRING statement, conditions that would cause an overflow condition are not encountered, then:

1. The transfer of data is completed.
2. The ON OVERFLOW phrase, if specified, is ignored.
3. Control is transferred to the end of the UNSTRING statement or, if the NOT ON OVERFLOW phrase is specified, to imperative-statement-2.

**imperative-statement-2**

Statement or statements for dealing with an overflow condition that does not occur.

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 according to 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.

## END-UNSTRING Phrase

This explicit scope terminator serves to delimit the scope of the UNSTRING statement. END-UNSTRING permits a conditional UNSTRING statement to be nested in another conditional statement. END-UNSTRING may also be used with an imperative UNSTRING statement.

For more information, see “Delimited Scope Statements” on page 247.

## UNSTRING Statement

## Data Flow

When the UNSTRING statement is initiated, data is transferred from the sending field to the current data receiving field, according to the following rules:

**Stage 1: Examine**

1. If the POINTER phrase is specified, the field is examined, beginning at the relative character position specified by the value in the pointer field.

   If the POINTER phrase is *not* specified, the sending field character-string is examined, beginning with the leftmost character.

2. If the DELIMITED BY phrase is specified, the examination proceeds from left to right, character-by-character, until a delimiter is encountered. If the end of the sending field is reached before a delimiter is found, the examination ends with the last character in the sending field. If there are more receiving fields, the next one is selected; otherwise, an overflow condition occurs.

   If the DELIMITED BY phrase is *not* specified, the number of characters examined is equal to the size of the current data receiving field, which depends on its data category, as shown in Table 37 on page 316.

*Table 45. Characters examined when DELIMITED BY is not specified*

| IF the receiving field is . . . | THEN the number of characters examined is . . . |
|---|---|
| alphanumeric or alphabetic | equal to the number of characters in the current receiving field |
| numeric | equal to the number of characters in integer portion of the current receiving field |
| described with the SIGN IS SEPARATE clause | 1 less than the size of the current receiving field |
| described as a variable-length data item | determined by the size of the current receiving field at the beginning of the UNSTRING operation |

**Stage 2: Move**

3. The examined characters (excluding any delimiter characters) are treated as an alphanumeric elementary item, and are moved into the current data receiving field, according to the rules for the MOVE statement (see "MOVE Statement" on page 333).
4. If the DELIMITER IN phrase is specified, the delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved to the current delimiter receiving field, according to the rules for the MOVE statement. If the delimiting condition is the end of the sending field, the current delimiter receiving field is filled with spaces.
5. If the COUNT IN phrase is specified, a value equal to the number of examined characters (excluding any delimiters) is moved into the data count field, according to the rules for an elementary move.

**Stage 3: Successive Iterations**

6. If the DELIMITED BY phrase is specified, the sending field is further examined, beginning with the first character to the right of the delimiter.

   If the DELIMITED BY phrase is *not* specified, the sending field is further examined, beginning with the first character to the right of the last character examined.

7. For each succeeding data receiving field, this process of examining and moving is repeated until either of the following occurs:

   - All the characters in the sending field have been transferred.
   - There are no more unfilled data receiving fields.

## Values at the End of Execution of the UNSTRING Statement

The following operations are performed only once, at the beginning of the execution of the UNSTRING statement:

- Calculations of subscripts, reference modifications, variable-lengths, variable locations
- Evaluations of functions

Therefore, if identifier-4, identifier-5, identifier-6, identifier-7, or identifier-8 is used as a subscript, reference-modifier, or function argument in the UNSTRING statement, or affects the length or location of any of the identifiers in the UNSTRING statement, then these values are determined at the beginning of the UNSTRING statement, and are *not* affected by any results of the UNSTRING statement.

## Example of the UNSTRING statement

Figure 19 shows the execution results for an example of the UNSTRING statement.

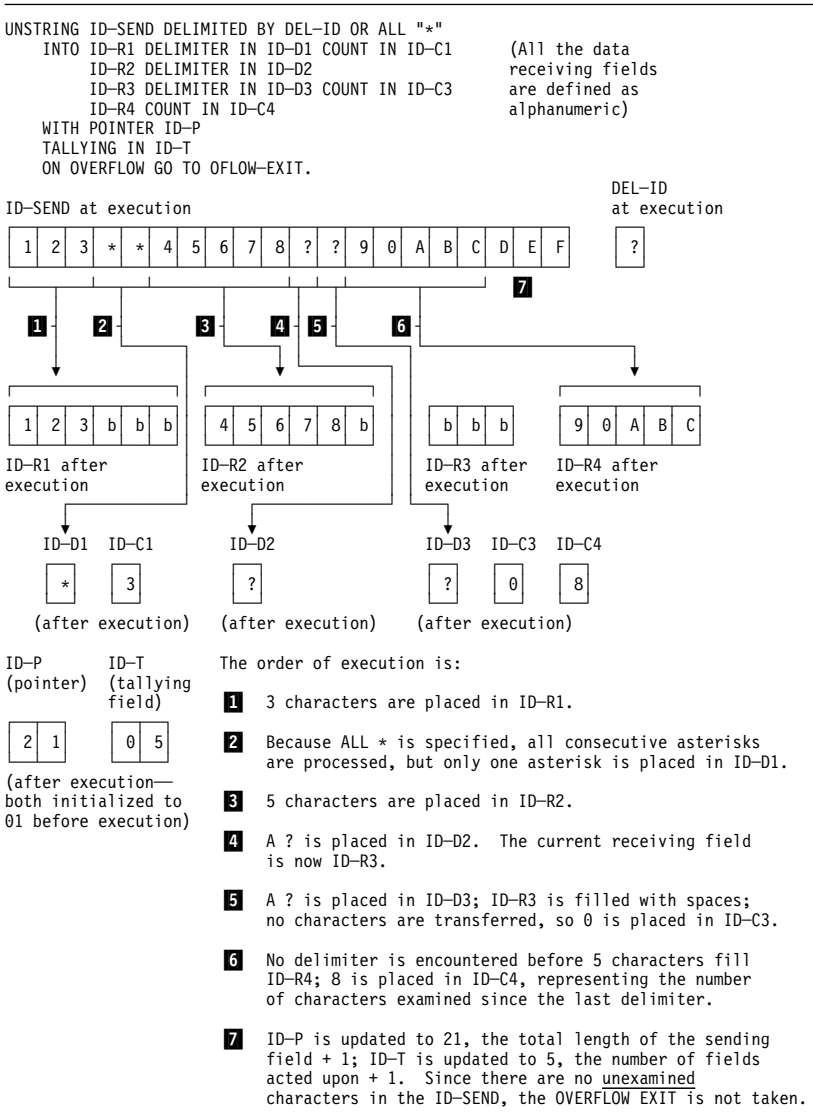



*Figure 19. Results of UNSTRING Statement Execution*

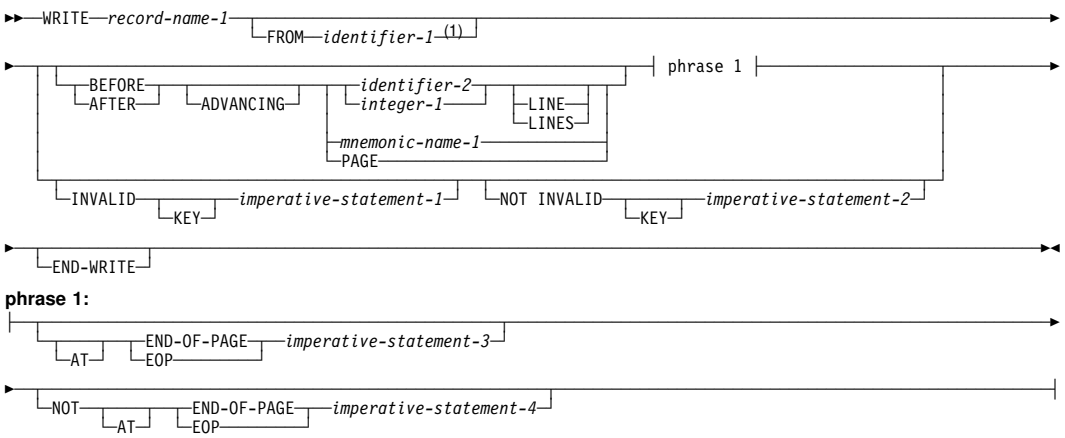
## WRITE Statement

The WRITE statement releases a logical record for an output or input/output file.

When the WRITE statement is executed:

- The associated sequential file must be open in OUTPUT or EXTEND mode.
- The associated indexed or relative file must be open in OUTPUT, I-O, or EXTEND mode.

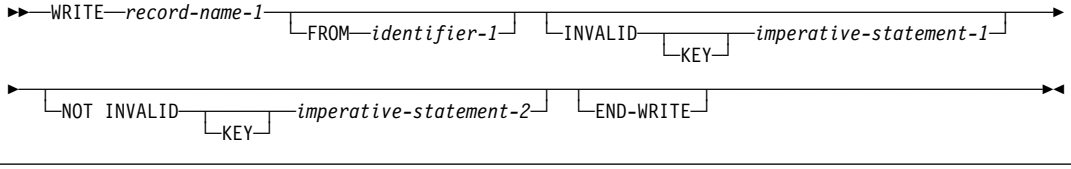
**Format 1—Sequential Files**



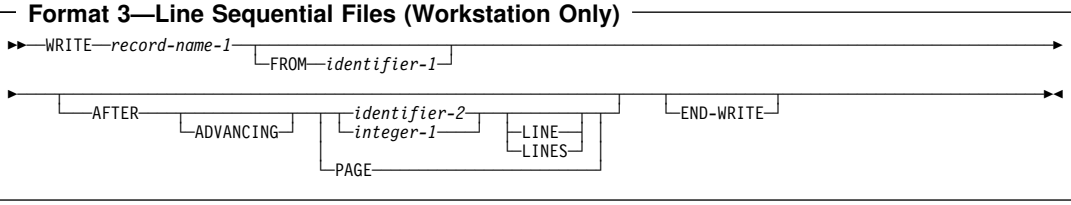

**Note:**

1 The BEFORE, AFTER, INVALID KEY, and AT END OF PAGE phrases are not valid for STL files, MVS VSAM files or OS/2 Btrieve files.

**Format 2—Indexed and Relative Files**




**Format 3—Line Sequential Files (Workstation Only)**



**record-name-1**

Must be defined in a Data Division FD entry. Record-name-1 can be qualified. It must not be associated with a sort or merge file.

x Can define a floating-point data item or DBCS data item. Identifier-1 must be a
x DBCS data item if record-name-1 is a DBCS data item.

x For relative files, as an IBM extension, the number of character positions in the
x record-name can be different from the number of character positions in the record
x being replaced.

**FROM phrase**

The result of the execution of the WRITE statement with the FROM identifier-1 phrase is equivalent to the execution of the following statements in the order specified:

```
MOVE identifier-1 TO record-name-1.
WRITE record-name-1.
```

The MOVE is performed according to the rules for the MOVE statement without the CORRESPONDING phrase.

**identifier-1**

Identifier-1 can be any of the following:

- The name of an entry in the Working-Storage Section or the LINKAGE SECTION
- The name of a record description for another previously opened file
- The name of an alphanumeric function identifier
- x A floating-point data item or a DBCS data item

Identifier-1 and record-name-1 must not refer to the same storage area.

After the WRITE statement is executed, the information is still available in identifier-1. (See “INTO/FROM Identifier Phrase” under “Common Processing Facilities” on page 254.)

**identifier-2**

Must be an integer data item.

## ADVANCING Phrase

The ADVANCING phrase controls positioning of the output record on the page.

WRKSTN Under AIX, OS/2, or Windows, when using WRITE ADVANCING with environment names C01-C012 or S01-S05, one line is advanced. For line sequential files, the BEFORE phrase is not supported.

Under OS/2, the BEFORE and AFTER phrases are not supported for Btrieve files.

MVS/VM Under MVS, the BEFORE and AFTER phrases are not supported for VSAM files. Under MVS, QSAM files are sequentially organized. The ADVANCING and END-OF-PAGE phrases control the vertical positioning of each line on a printed page. As an IBM extension, you can specify the ADVANCING PAGE and END-OF-PAGE phrases in a single WRITE statement.

If the printed page is held on an intermediate device (a disk, for example), the format can appear different than the expected output when it is edited or browsed.

### ADVANCING Phrase Rules

When the ADVANCING phrase is specified, the following rules apply:

1. When BEFORE ADVANCING is specified, the line is printed before the page is advanced.
2. When AFTER ADVANCING is specified, the page is advanced before the line is printed.
3. When identifier-2 is specified, the page is advanced the number of lines equal to the current value in identifier-2. Identifier-2 must name an elementary integer data item.
4. When integer is specified, the page is advanced the number of lines equal to the value of integer.
5. Integer or the value in identifier-2 can be zero.
6. When PAGE is specified, the record is printed on the logical page BEFORE or AFTER (depending on the phrase used) the device is positioned to the next logical page. If PAGE has no meaning for the device used, then BEFORE or AFTER (depending on the phrase specified) ADVANCING 1 LINE is provided.

   If the FD entry contains a LINAGE clause, the repositioning is to the first printable line of the next page, as specified in that clause. If the LINAGE clause is omitted, the repositioning is to line 1 of the next succeeding page.
7. MVS/VM When mnemonic-name is specified, a skip to channels 1 through 12, or space suppression, takes place. Mnemonic-name must be equated with environment-name-1 in the SPECIAL-NAMES paragraph.

   The mnemonic-name phrase can also be specified for stacker selection with a card punch file. When using stacker selection, WRITE AFTER ADVANCING must be used.

MVS/VM The ADVANCING phrase of the WRITE statement, or the presence of a LINAGE clause on the file, causes a carriage control character to be generated in the record that is written. If the corresponding file connector is EXTERNAL, all file connectors within the run unit must be defined such that carriage control characters will be generated for records that are written. That is, if all the files have a LINAGE clause, some of the programs can use the WRITE statement with the ADVANCING phrase and other programs can use the WRITE statement without the ADVANCING phrase. However, if none of the files has a LINAGE clause, then if any of the programs use the WRITE statement with the ADVANCING phrase, all of the programs in the run unit that have a WRITE statement must use the WRITE statement with the ADVANCING phrase.

### LINAGE-COUNTER Rules

If the LINAGE clause is specified for this file, the associated LINAGE-COUNTER special register is modified during the execution of the WRITE statement, according to the following rules:

1. If ADVANCING PAGE is specified, LINAGE-COUNTER is reset to 1.
2. If ADVANCING identifier-2 or integer is specified, LINAGE-COUNTER is increased by the value in identifier-2 or integer.
3. If the ADVANCING phrase is omitted, LINAGE-COUNTER is increased by 1.
4. When the device is repositioned to the first available line of a new page, LINAGE-COUNTER is reset to 1.

When this phrase is omitted, automatic line advancing is provided, as if the user had written AFTER ADVANCING 1 LINE.

**Note:** MVS/VM If you use the ADV compiler option, the compiler adds 1 byte to the record length in order to allow for the control character. If in your record definition you already reserve the first byte for the control character, you should use the NOADV option. For files defined with the LINAGE clause, the NOADV option has no effect. The compiler processes these files as if the ADV option were specified.

## END-OF-PAGE Phrases

x WRKSTN Under AIX, OS/2, or Windows, the END-OF-PAGE phrase is not supported
x for line sequential files or, on OS/2, Btrieve files.

MVS/VM Under MVS, the AT END-OF-PAGE phrase is not supported for VSAM files.

When END-OF-PAGE is specified, and the logical end of the printed page is reached during execution of the WRITE statement, the END-OF-PAGE imperative-statement is executed. When the END-OF-PAGE phrase is specified, the FD entry for this file must contain a LINAGE clause.

The logical end of the printed page is specified in the associated LINAGE clause.

An END-OF-PAGE condition is reached when execution of a WRITE END-OF-PAGE statement causes printing or spacing within the footing area of a page body. This occurs when execution of such a WRITE statement causes the value in the LINAGE-COUNTER special register to equal or exceed the value specified in the WITH FOOTING phrase of the LINAGE clause. The WRITE statement is executed, and then the END-OF-PAGE imperative-statement is executed.

An automatic page overflow condition is reached whenever the execution of any given WRITE statement (with or without the END-OF-PAGE phrase) cannot be completely executed within the current page body. This occurs when a WRITE statement, if executed, would cause the value in the LINAGE-COUNTER to exceed the number of lines for the page body specified in the LINAGE clause. In this case, the line is printed BEFORE or AFTER (depending on the option specified) the device is repositioned to the first printable line on the next logical page, as specified in the LINAGE clause. If the END-OF-PAGE phrase is specified, the END-OF-PAGE imperative-statement is then executed.

If the WITH FOOTING phrase of the LINAGE clause is not specified, the automatic page overflow condition exists because no end-of-page condition (as distinct from the page overflow condition) can be detected.

If the WITH FOOTING phrase is specified, but the execution of a given WRITE statement would cause the LINAGE-COUNTER to exceed both the footing value and the page body value specified in the LINAGE clause, then both the end-of-page condition and the automatic page overflow condition occur simultaneously.

The key words END-OF-PAGE and EOP are equivalent.

x As an IBM extension, you can specify both the ADVANCING PAGE and
x END-OF-PAGE phrases in a single WRITE statement.

## INVALID KEY Phrases

x WRKSTN Under AIX, OS/2, or Windows, the INVALID KEY phrase is not supported for
x line sequential files or, for OS/2, Btrieve files.

x MVS/VM Under MVS, the INVALID KEY phrase is not supported for VSAM sequential
x files.

An invalid key condition is caused by the following:

x • **For sequential files**:
x – An attempt is made to write beyond the externally defined boundary of the file.

• **For indexed files**:
– An attempt is made to write beyond the externally defined boundary of the file.
– ACCESS SEQUENTIAL is specified and the file is opened OUTPUT, and the value of the prime record key is not greater than that of the previous record.
– The file is opened OUTPUT or I-O and the value of the prime record key equals that of an already existing record.

• **For relative files**:
– An attempt is made to write beyond the externally defined boundary of the file.
– When the access mode is random or dynamic and the RELATIVE KEY data item specifies a record that already exists in the file
– The number of significant digits in the relative record number is larger than the size of the relative key data item for the file.

When an invalid key condition occurs:

• If the INVALID KEY phrase is specified, imperative-statement-1 is executed. (See Table 32 on page 255).

• Otherwise, the WRITE statement is unsuccessful and the contents of record-name are unaffected (except for MVS QSAM files). And, the following occurs:

x – **For sequential files**—the status key, if specified, is updated and an
x EXCEPTION/ERROR condition exists.

x If an explicit or implicit EXCEPTION/ERROR procedure is specified for the file,
x the procedure is executed. If no such procedure is specified, the results are
x unpredictable.

– **For relative and indexed files**—program execution proceeds according to the rules described under “Invalid key condition” on page 256.

The INVALID KEY conditions that apply to a relative file in OPEN OUTPUT mode also apply to one in OPEN EXTEND mode.

- If the NOT INVALID KEY phrase is specified and a valid key condition exists at the end of the execution of the WRITE statement, control is passed to imperative-statement-4.

x As an IBM extension, you can omit both the INVALID KEY phrase and the
x EXCEPTION/ERROR procedure.

### END-WRITE Phrase

This explicit scope terminator serves to delimit the scope of the WRITE statement. END-WRITE permits a conditional WRITE statement to be nested in another conditional statement. END-WRITE can also be used with an imperative WRITE statement.

For more information, see “Delimited Scope Statements” on page 247.

## x Effect of Record Locking (LOCK MODE Clause)

x WRKSTN Under AIX, OS/2, and Windows, the following applies to files using record
x locking:

- x If two or more file connectors for a sequential file add records by sharing the file
x after opening it in EXTEND mode, the records are in an unspecified order.
- x If two or more file connectors for a relative file add records by sharing the file after
x opening it in EXTEND mode, the relative key values returned are ascending, but
x not necessary consecutive.
- x If two or more file connectors for an indexed file add records by sharing the file
x after opening it in EXTEND mode, the order of alternate keys allowing for dupli-
x cates is unspecified.
- x If the LOCK MODE IS AUTOMATIC is specified, a successful WRITE statement
x releases an existing record lock.

x

### WRITE for Sequential Files

The maximum record size for the file is established at the time the file is created, and cannot subsequently be changed.

After the WRITE statement is executed, the logical record is no longer available in record-name-1, unless:

- The associated file is named in a SAME RECORD AREA clause (in which case, the record is also available as a record of the other files named in the SAME RECORD AREA clause), or
- The WRITE statement is unsuccessful because of a boundary violation.

In either of these two cases, the logical record is still available in record-name-1.

The file position indicator is not affected by execution of the WRITE statement.

The number of character positions required to store the record in a file might or might not be the same as the number of character positions defined by the logical description of that record in the COBOL program. (See “PICTURE Clause Editing” on page 180 and “USAGE Clause” on page 200.)

If the FILE STATUS clause is specified in the File-Control entry, the associated status key is updated when the WRITE statement is executed, whether or not execution is successful.

The WRITE statement can only be executed for a sequential file opened in OUTPUT or EXTEND mode for QSAM files.

### Multivolume Files

MVS/VM When end-of-volume is recognized for a multivolume OUTPUT file (tape or sequential direct-access file), the WRITE statement performs the following operations:

- The standard ending volume label procedure
- A volume switch
- The standard beginning volume label procedure

### Punch Function Files with the IBM 3525

MVS/VM When the punch function is used, the next I-O operation after the READ statement must be a WRITE statement for the punch function file.

If you want to punch additional data into some of the cards and not into others, a dummy WRITE statement must be issued for the null cards, first filling the output area with SPACES.

x If stacker selection for the punch function file is desired, you can specify the appropriate
x stacker function-names in the SPECIAL-NAMES paragraph, and then issue WRITE
x ADVANCING statements using the associated mnemonic-names.

### Print Function Files

MVS/VM After the punch function operations (if specified) are completed, you can issue WRITE statement(s) for the print function file.

If you wish to print additional data on some of the data cards and not on others, the WRITE statement for the null cards can be omitted. Any attempt to write beyond the limits of the card results in abnormal termination of the job; thus, the END-OF-PAGE phrase cannot be specified.

Depending on the capabilities of the specific IBM 3525 model in use, the print file can be either a 2-line print file or a multiline print file. Up to 64 characters can be printed on each line.

- For a 2-line print file, the lines are printed on line 1 (top edge of card) and line 3 (between rows 11 and 12). Line control cannot be specified. Automatic spacing is provided.
- For a multiline print file, up to 25 lines of characters can be printed. Line control can be specified. If line control is not specified, automatic spacing is provided.

Line control is specified by issuing WRITE AFTER ADVANCING statements for the print function file. If line control is used for one such statement, it must be used for all other WRITE statements issued to the file. The maximum number of printable characters, including any space characters, is 64. Such WRITE statements must not specify space suppression.

Identifier and integer have the same meanings they have for other WRITE AFTER ADVANCING statements. However, such WRITE statements must not increase the line position on the card beyond the card limit, or abnormal termination results.

The mnemonic-name option of the WRITE AFTER ADVANCING statement can also be specified. In the SPECIAL-NAMES paragraph, the environment-names can be associated with the mnemonic-names, as follows:

*Table 46. Meanings of Environment-names in SPECIAL NAMES paragraph*

| Environment-name | Meaning |
|---|---|
| **C02** | **Line 3** |
| **C03** | **Line 5** |
| **C04** | **Line 7** |
| . | . |
| . | . |
| . | . |
| **C12** | **Line 23** |

## Advanced Function Printing

MVS/VM When using the WRITE ADVANCING phrase with a mnemonic-name associated with environment-name AFP-5A, a Print Services Facility (PSF) control character is placed in the control character position of the output record. This control character (X'5A') allows Advanced Function Printing (AFP) services to be used. For more information, refer to the documentation for the Print Services Facility products: PSF/MVS (5695-040), PSF/VM (5684-141), or PSF/VSE (5686-040).

## WRITE Statement

### WRITE for Indexed Files

Before the WRITE statement is executed, you must set the prime record key (the RECORD KEY data item, as defined in the File-Control entry) to the desired value. (Note that RECORD KEY values must be unique within a file.)

If the ALTERNATE RECORD KEY clause is also specified in the File-Control entry, each alternate record key must be unique, unless the DUPLICATES phrase is specified. If the DUPLICATES phrase is specified, alternate record key values might not be unique. In this case, the system stores the records so that later sequential access to the records allows retrieval in the same order in which they were stored.

When ACCESS IS SEQUENTIAL is specified in the File-Control entry, records must be released in ascending order of RECORD KEY values.

When ACCESS is RANDOM or ACCESS IS DYNAMIC is specified in the File-Control entry, records may be released in any programmer-specified order.

### WRITE for Relative Files

For OUTPUT files, the WRITE statement causes the following actions:

- If ACCESS IS SEQUENTIAL is specified:

  The first record released has relative record number 1, the second record released has relative record number 2, the third number 3, and so on.

  If the RELATIVE KEY is specified in the File-Control entry, the relative record number of the record just released is placed in the RELATIVE KEY during execution of the WRITE statement.

- If ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, the RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, this record is placed at the specified relative record number position in the file.

For I-O files, either ACCESS IS RANDOM or ACCESS IS DYNAMIC must be specified; the WRITE statement inserts new records into the file. The RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, this record is placed at the specified relative record number position in the file.

# Part 7. Intrinsic Functions


**Intrinsic Functions** . . . . . 426
Specifying a Function . . . . . 426
Function Definitions . . . . . 432
ACOS . . . . . 435
ANNUITY . . . . . 436
ASIN . . . . . 437
ATAN . . . . . 438
CHAR . . . . . 439
COS . . . . . 440
CURRENT-DATE . . . . . 441
DATE-OF-INTEGER . . . . . 443
DAY-OF-INTEGER . . . . . 444
FACTORIAL . . . . . 445
INTEGER . . . . . 446
INTEGER-OF-DATE . . . . . 447
INTEGER-OF-DAY . . . . . 448
INTEGER-PART . . . . . 449
LENGTH . . . . . 450
LOG . . . . . 451
LOG10 . . . . . 452
LOWER-CASE . . . . . 453
MAX . . . . . 454
MEAN . . . . . 455
MEDIAN . . . . . 456
MIDRANGE . . . . . 457
MIN . . . . . 458
MOD . . . . . 459
NUMVAL . . . . . 460
NUMVAL-C . . . . . 461
ORD . . . . . 463
ORD-MAX . . . . . 464
ORD-MIN . . . . . 465
PRESENT-VALUE . . . . . 466
RANDOM . . . . . 467
RANGE . . . . . 468
REM . . . . . 469
REVERSE . . . . . 470
SIN . . . . . 471
SQRT . . . . . 472
STANDARD-DEVIATION . . . . . 473
SUM . . . . . 474
TAN . . . . . 475
UPPER-CASE . . . . . 476
VARIANCE . . . . . 477
WHEN-COMPILED . . . . . 478


© Copyright IBM Corp. 1991, 1996

# Intrinsic Functions

Data processing problems often require the use of values that are not directly accessible in the data storage associated with the object program, but instead must be derived through performing operations on other data. An intrinsic function is a function that performs a mathematical, character, or logical operation, and thereby allows you to make reference to a data item whose value is derived automatically during the execution of the object program.

The functions can be grouped into six categories, based on the type of service performed: mathematical, statistical, date/time, financial, character-handling, and general.

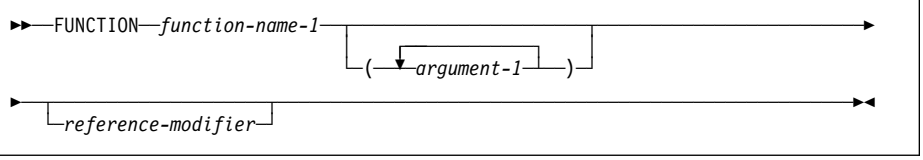
You can reference a function by specifying its name, along with any required arguments, in a Procedure Division statement.

Functions are elementary data items, and return alphanumeric, numeric or integer values. Functions cannot serve as receiving operands.

## Specifying a Function

The general format of a function-identifier is:

**Format**




**function-name-1**
Function-name-1 must be one of the Intrinsic Function names.

**argument-1**
Argument-1 must be an identifier, literal (other than a figurative constant), or arithmetic expression.

**reference-modifier**
Can be specified only for functions of the category alphanumeric

Below, we will show examples of an intrinsic function invocation for an alphanumeric source statement and a numeric source statement.

The alphanumeric source statement:

```
MOVE FUNCTION UPPER-CASE("hello") TO DATA-NAME.
```

replaces each lowercase letter in the argument with the corresponding uppercase letter, resulting in the movement of HELLO into DATA-NAME.


© Copyright IBM Corp. 1991, 1996

The numeric source statement,

```
COMPUTE NUM-ITEM = FUNCTION SUM(A B C)
```

Adds the values of A, B, and C and places the result in NUM-ITEM.

Within a Procedure Division statement, each function-identifier is evaluated at the same time as any reference modification or subscripting associated with an identifier in that same position would be evaluated.

## Function Definition and Evaluation

The class and characteristics of a function, and the number and types of arguments it requires, are determined by its function definition. These characteristics include:

- For some functions, the class and characteristics are determined by the arguments to the function
- For alphanumeric functions, the size of the returned value
- For numeric and integer functions, the sign of the returned value, and whether the function is integer
- The actual value returned by the function

The evaluation of any intrinsic function is not affected by the context in which it appears; in other words, function evaluation is not affected by operations or operands outside the function. However, evaluation of a function can be affected by the attributes of its arguments.

## Types of Functions

There are three types of functions:

- Alphanumeric
- Numeric
- Integer

**Alphanumeric** functions are of the class and category alphanumeric. The value returned has an implicit usage of DISPLAY and is in standard data format characters. The number of character positions in the value returned is determined by the function definition.

**Numeric** functions are of the class and category numeric. The returned value is always considered to have an operational sign and is a numeric intermediate result. For more information, see the *IBM COBOL Programming Guide* for your platform.

**Integer** functions are of the class and category numeric. The returned value is always considered to have an operational sign and is an integer intermediate result. The number of digit positions in the value returned is determined by the function definition. For more information, see the *IBM COBOL Programming Guide* for your platform.

## Intrinsic Functions

## Rules for Usage

**Alphanumeric Functions**

An alphanumeric function can be specified anywhere in the general formats that an identifier is permitted and where the rules associated with the general formats do not specifically prohibit reference to functions, except as follows:

- As a receiving operand of any statement
- Where the rules associated with the general formats require the data item being referenced to have particular characteristics (such as class and category, usage, size, and permissible values) and the evaluation of the function according to its definition and the particular arguments specified would not have these characteristics.

A reference modification for an alphanumeric function is allowed. If reference modification is specified for a function, the evaluation of the reference modification takes place immediately after the evaluation of the function.

An alphanumeric function can be referenced as an argument for a function which allows an alphanumeric argument.

**Numeric Functions**

A numeric function can be used only where an arithmetic expression can be specified.

A numeric function can be referenced as an argument for a function which allows a numeric argument.

A numeric function cannot be used where an integer operand is required, even if the particular reference will yield an integer value. The INTEGER or INTEGER-PART functions can be used to force the type of a numeric argument to be an integer.

**Integer Functions**

An integer function can be used only where an arithmetic expression can be specified.

An integer function can be referenced as an argument for a function which allows an integer argument.

**Special Usage Notes:**

Identifier-2 of the CALL statement must not be a function-identifier.

The COPY statement will allow function-identifiers of all types in the REPLACING phrase.

## Arguments

The values returned by some functions are determined by the arguments specified in the function-identifier when the functions are evaluated. Some functions require no arguments; others require a fixed number of arguments, and still others allow a variable number of arguments.

An argument must be one of the following:

- An identifier
- An arithmetic expression
- A function-identifier
- A literal other than a figurative constant.
- A special-register

The argument to a function can be any function or an expression containing a function, including another evaluation of the same function, whose result meets the category requirement for the argument.

x An argument cannot be a DBCS literal or data item. See “Function Definitions” on
x page 432 for function specific argument specifications.

The types of arguments are:

- **Alphabetic.** An elementary data item of the class alphabetic or a nonnumeric literal containing only alphabetic characters. The content of the argument will be used to determine the value of the function. The length of the argument can be used to determine the value of the function.
- **Alphanumeric.** A data item of the class alphabetic or alphanumeric or a nonnumeric literal. The content of the argument will be used to determine the value of the function. The length of the argument can be used to determine the value of the function.
- **Integer.** An arithmetic expression that will always result in an integer value. The value of this expression, including its sign, is used to determine the value of the function.
- **Numeric.** An arithmetic expression, whose value, including its sign, is used to determine the value of the function.

Some functions place constraints on their arguments, such as the range of values acceptable. If the values assigned as arguments for a function do not comply with specified constraints, the returned value is undefined.

If a nested function is used as an argument, the evaluation of its arguments will not be affected by the arguments in the outer function.

Only those arguments at the same function level interact with each other. This interaction occurs in two areas:

- The computation of an arithmetic expression that appears as a function argument will be affected by other arguments for that function.
- The evaluation of the function takes into consideration the attributes of all of its arguments.

When a function is evaluated, its arguments are evaluated individually in the order specified in the list of arguments, from left to right. The argument being evaluated can be a function-identifier, or it can be an expression containing function-identifiers.

If an arithmetic expression is specified as an argument, and if the first operator in the expression is a unary plus or a unary minus, it must be immediately preceded by a left parenthesis.

x Floating-point literals are allowed wherever a numeric argument is allowed, and in arithx metic expressions used in functions that allow a numeric argument. They are not x allowed where an integer argument is required.

x External floating-point items are allowed wherever a numeric argument is allowed, and x in arithmetic expressions used in functions that allow a numeric argument.

x External floating-point items are not allowed where an integer argument is required, or x where an argument of alphanumeric class is allowed in a function identification, such as x in the LOWER-CASE, REVERSE, UPPER-CASE, NUMVAL, and NUMVAL-C functions.

## ALL Subscripting

When a function allows an argument to be repeated a variable number of times, you can refer to a table by specifying the data-name and any qualifiers that identify the table. This can be followed immediately by subscripting where one or more of the subscripts is the word ALL.

**Note:** The evaluation of an ALL subscript must result in at least one argument or the value returned by the function will be undefined; however, the situation can be diagnosed at run-time by specifying the SSRANGE compiler option and the CHECK runtime option.

Specifying ALL as a subscript is equivalent to specifying all table elements possible using every valid subscript in that subscript position.

For a table argument specified as "Table-name(ALL)", the order of the implicit specification of each table element as an argument is from left to right, where the first (or leftmost) argument is "Table-name(1)" and ALL has been replaced by 1. The next argument is "Table-name(2)", where the subscript has been incremented by 1. This process continues, with the subscript being incremented by 1 to produce an implicit argument, until the ALL subscript has been incremented through its range of values.

For example,

```
FUNCTION MAX(Table(ALL))
```

is equivalent to

```
FUNCTION MAX(Table(1) Table(2) Table(3)... Table(n))
```

where n is the number of elements in Table.

If there are multiple ALL subscripts, "Table-name(ALL, ALL, ALL)", the first implicit argument is "Table-name(1, 1, 1)", where each ALL has been replaced by 1. The next argument is "Table-name(1, 1, 2)", where the rightmost subscript has been incremented by 1. The subscript represented by the rightmost ALL is incremented through its range of values to produce an implicit argument for each value.

Once a subscript specified as ALL has been incremented through its range of values, the next subscript to the left that is specified as ALL is incremented by 1. Each subscript specified as ALL to the right of the newly incremented subscript is set to 1 to produce an implicit argument. Once again, the subscript represented by the rightmost ALL is incremented through its range of values to produce an implicit argument for each value. This process is repeated until each subscript specified as ALL has been incremented through its range of values.

For example,

```
FUNCTION MAX(Table(ALL, ALL))
```

is equivalent to

```
FUNCTION MAX(Table(1, 1) Table(1, 2) Table(1, 3)... Table(1, n)
             Table(2, 1) Table(2, 2) Table(2, 3)... Table(2, n)
             Table(3, 1) Table(3, 2) Table(3, 3)... Table(3, n)
             .
             .
             .
             Table(m, 1) Table(m, 2) Table(m, 3)... Table(m, n))
```

where n is the number of elements in the column dimension of Table, and m is the number of elements in the row dimension of Table.

ALL subscripts can be combined with literal, data-name, or index-name subscripts to reference multidimensional tables.

For example,

```
FUNCTION MAX(Table(ALL, 2))
```

is equivalent to

```
FUNCTION MAX(Table(1, 2)
             Table(2, 2)
             Table(3, 2)
             .
             .
             .
             Table(m, 2))
```

where m is the number of elements in the row dimension of Table.

If an ALL subscript is specified for an argument and the argument is reference modified, then the reference-modifier is applied to each of the implicitly specified elements of the table.

If an ALL subscript is specified for an operand that is reference-modified, the reference-modifier is applied to each of the implicitly specified elements of the table.

If the ALL subscript is associated with an OCCURS DEPENDING ON clause, the range of values is determined by the object of the OCCURS DEPENDING ON clause.

For example, given a payroll record definition such as:

```
01 PAYROLL.
 02 PAYROLL-WEEK   PIC 99.
 02 PAYROLL-HOURS  PIC 999 OCCURS 1 TO 52
    DEPENDING ON PAYROLL-WEEK.
```

The following COMPUTE statements could be used to identify total year-to-date hours, the maximum hours worked in any week, and the specific week corresponding to the maximum hours:

```
COMPUTE YTD-HOURS = FUNCTION SUM (PAYROLL-HOURS(ALL))
COMPUTE MAX-HOURS = FUNCTION MAX (PAYROLL-HOURS(ALL))
COMPUTE MAX-WEEK  = FUNCTION ORD-MAX (PAYROLL-HOURS(ALL))
```

In these function invocations the subscript ALL is used to reference all elements of the PAYROLL-HOURS array (depending on the execution time value of the PAYROLL-WEEK field).

## Function Definitions

Table 47 provides an overview of the argument type, function type and value returned for each of the intrinsic functions. Argument types and function types are abbreviated as follows:

- A = alphabetic
- I = integer
- N = numeric
- X = alphanumeric

*Table 47 (Page 1 of 3). Table of Functions*

| FUNCTION-NAME | ARGUMENTS | TYPE | VALUE RETURNED |
|---|---|---|---|
| ACOS | N1 | N | Arccosine of N1 |
| ANNUITY | N1, I2 | N | Ratio of annuity paid for I2 periods at interest of N1 to initial investment of one |
| ASIN | N1 | N | Arcsine of N1 |
| ATAN | N1 | N | Arctangent of N1 |
| CHAR | I1 | X | Character in position I1 of program collating sequence |
| COS | N1 | N | Cosine of N1 |
| CURRENT-DATE | None | X | Current date and time and difference from Greenwich Mean Time |
| DATE-OF-INTEGER | I1 | I | Standard date equivalent (YYYYMMDD) of integer date |
| DAY-OF-INTEGER | I1 | I | Julian date equivalent (YYYYDDD) of integer date |

*Table 47 (Page 2 of 3). Table of Functions*

| FUNCTION-NAME | ARGUMENTS | TYPE | VALUE RETURNED |
|---|---|---|---|
| FACTORIAL | I1 | I | Factorial of I1 |
| INTEGER | N1 | I | The greatest integer not greater than N1 |
| INTEGER-OF-DATE | I1 | I | Integer date equivalent of standard date (YYYYMMDD) |
| INTEGER-OF-DAY | I1 | I | Integer date equivalent of Julian date (YYYYDDD) |
| INTEGER-PART | N1 | I | Integer part of N1 |
| LENGTH | A1, N1, or X1 | I | Length of argument |
| LOG | N1 | N | Natural logarithm of N1 |
| LOG10 | N1 | N | Logarithm to base 10 of N1 |
| LOWER-CASE | A1 or X1 | X | All letters in the argument are set to lowercase |
| MAX | A1... or | X | Value of maximum argument; note that the type of function depends on the arguments |
| | I1... or | I | |
| | N1... or | N | |
| | X1... | X | |
| MEAN | N1... | N | Arithmetic mean of arguments |
| MEDIAN | N1... | N | Median of arguments |
| MIDRANGE | N1... | N | Mean of minimum and maximum arguments |
| MIN | A1... or | X | Value of minimum argument; note that the type of function depends on the arguments |
| | I1... or | I | |
| | N1... or | N | |
| | X1... | X | |
| MOD | I1,I2 | I | I1 modulo I2 |
| NUMVAL | X1 | N | Numeric value of simple numeric string |
| NUMVAL-C | X1 or<br>X1,X2 | N | Numeric value of numeric string with optional commas and currency sign |
| ORD | A1 or X1 | I | Ordinal position of the argument in collating sequence |
| ORD-MAX | A1..., N1..., or X1... | I | Ordinal position of maximum argument |
| ORD-MIN | A1..., N1..., or X1... | I | Ordinal position of minimum argument |

*Table 47 (Page 3 of 3). Table of Functions*

| FUNCTION-NAME | ARGUMENTS | TYPE | VALUE RETURNED |
|---|---|---|---|
| PRESENT-VALUE | N1 or N2... | N | Present value of a series of future period-end amounts, N2, at a discount rate of N1 |
| RANDOM | I1, none | N | Random number |
| RANGE | I1... or | I | Value of maximum argument minus value of minimum argument; note that the type of function depends on the arguments. |
| | N1... | N | |
| REM | N1,N2 | N | Remainder of N1/N2 |
| REVERSE | A1 or X1 | X | Reverse order of the characters of the argument |
| SIN | N1 | N | Sine of N1 |
| SQRT | N1 | N | Square root of N1 |
| STANDARD-DEVIATION | N1... | N | Standard deviation of arguments |
| SUM | I1... or | I | Sum of arguments; note that the type of function depends on the arguments. |
| | N1... | N | |
| TAN | N1 | N | Tangent of N1 |
| UPPER-CASE | A1 or X1 | X | All letters in the argument are set to uppercase |
| VARIANCE | N1... | N | Variance of arguments |
| WHEN-COMPILED | None | X | Date and time when program was compiled |

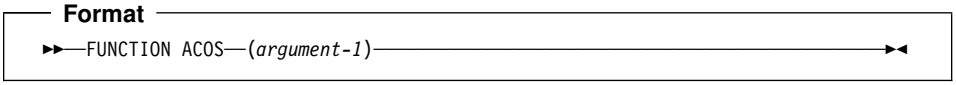
The following are the function definitions for each of the Intrinsic Functions summarized in Table 47 on page 432:

## ACOS

The ACOS function returns a numeric value in radians that approximates the arccosine of the argument specified.

The function type is numeric.

**Format**




**argument-1**

Must be class numeric. The value of argument-1 must be greater than or equal to -1 and less than or equal to +1.

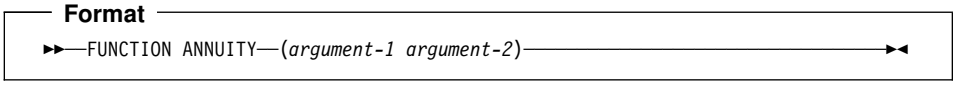
The returned value is the approximation of the arccosine of the argument and is greater than or equal to zero and less than or equal to Pi.

## ANNUITY

The ANNUITY function returns a numeric value that approximates the ratio of an annuity paid at the end of each period, for a given number of periods, at a given interest rate, to an initial value of one. The number of periods is specified by argument-2; the rate of interest is specified by argument-1. For example, if argument-1 is zero and argument-2 is four, the value returned is the approximation of the ratio 1 / 4.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric. The value of argument-1 must be greater than or equal to zero.

**argument-2**
Must be a positive integer.

When the value of argument-1 is zero, the value returned by the function is the approximation of:`1 / ARGUMENT-2`

When the value of argument-1 is not zero, the value of the function is the approximation of:

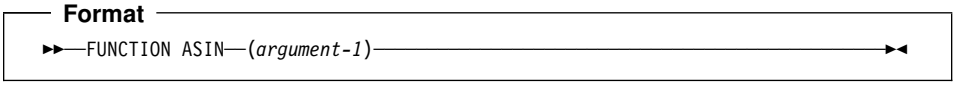
```
ARGUMENT-1 / (1 - (1 + ARGUMENT-1) ** (- ARGUMENT-2))
```

## ASIN

The ASIN function returns a numeric value in radians that approximates the arcsine of the argument specified.

The function type is numeric.

**Format**




**argument-1**

Must be class numeric. The value of argument-1 must be greater than or equal to -1 and less than or equal to +1.

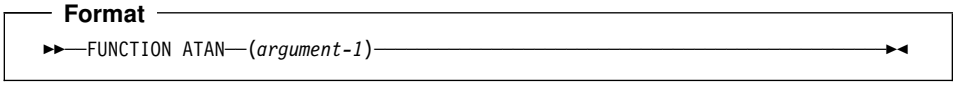
The returned value is the approximation of the arcsine of argument-1 and is greater than or equal to -Pi/2 and less than or equal to +Pi/2.

## ATAN

The ATAN function returns a numeric value in radians that approximates the arctangent of the argument specified.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric.

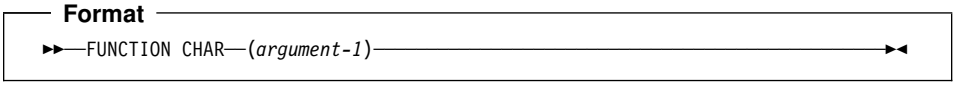
The returned value is the approximation of the arctangent of argument-1 and is greater than -Pi/2 and less than +Pi/2.

## CHAR

The CHAR function returns a one-character alphanumeric value that is a character in the program collating sequence having the ordinal position equal to the value of the argument specified.

The function type is alphanumeric.

**Format**




**argument-1**
Must be an integer. The value must be greater than zero and less than or equal to the number of positions in the collating sequence.

If more than one character has the same position in the program collating sequence, the character returned as the function value is that of the first literal specified for that character position in the ALPHABET clause.

If the current program collating sequence was not specified by an ALPHABET clause,

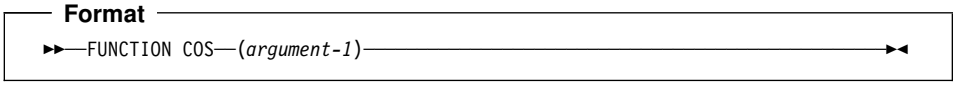
- Under MVS and VM, the EBCDIC collating sequence is used. (See Appendix B, “EBCDIC and ASCII Collating Sequences” on page 522.)
- Under AIX, OS2, and Windows, the COLLSEQ compiler option indicates the collating sequence used. For example, if COLLSEQ(EBCDIC) is specified and the PROGRAM COLLATING SEQUENCE is not specified (or is NATIVE), the EBCDIC collating sequence is applied.

## COS

The COS function returns a numeric value that approximates the cosine of the angle or arc specified by the argument in radians.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric.

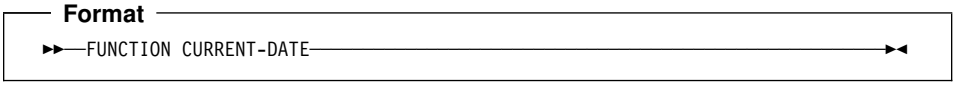
The returned value is the approximation of the cosine of the argument and is greater than or equal to -1 and less than or equal to +1.

## CURRENT-DATE

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date, time of day, and time differential from Greenwich Mean Time provided by the system on which the function is evaluated.

The function type is alphanumeric.

**Format**




Reading from left to right, the 21 character positions in the value returned can be interpreted as follows:

| 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 digits of the seconds past the minute, in the range 00 through 59. |
| **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 '-' or the character '+'. The character '-' is returned if the local time indicated in the previous character positions is behind Greenwich Mean Time. The character '+' is returned if the local time indicated is the same as or ahead of Greenwich Mean 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 reported time is behind Greenwich Mean Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich Mean 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 reported time is ahead of or behind Greenwich Mean Time, depending on whether character position 17 is '+' or '-', respectively. If character position 17 is '0', the value 00 is returned. |

## CURRENT-DATE

For more information, see the *IBM COBOL Programming Guide* for your platform.

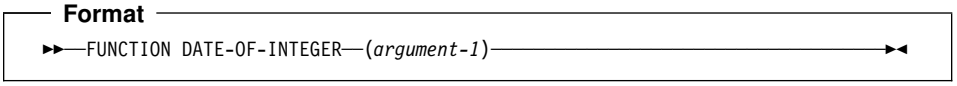
## DATE-OF-INTEGER

The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD).

The function type is integer.

The function result is an eight-digit integer.

**Format**




**argument-1**

A positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar. The valid range is 1 to 3,067,671, which corresponds to dates ranging from January 1, 1601 thru December 31, 9999.

MVS/VM Under MVS and VM, the INTDATE installation option affects the starting date for the integer date functions. For details, see the *IBM COBOL for MVS & VM Programming Guide*.

The returned value represents the International Standards Organization (ISO) standard date equivalent to the integer specified as argument-1.

The returned value is an integer of 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.

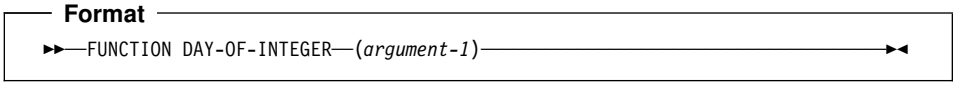
## DAY-OF-INTEGER

The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD).

The function type is integer.

The function result is a seven-digit integer.

**Format**




**argument-1**

A positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar. The valid range is 1 to 3,067,671, which corresponds to dates ranging from January 1, 1601 thru December 31, 9999.

x MVS/VM Under MVS and VM, the INTDATE installation option affects the starting
x date for the integer date functions. For details, see the *IBM COBOL for MVS &*
x *VM Programming Guide*.

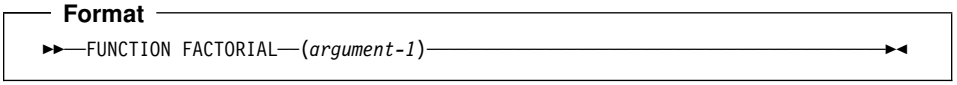
The returned value represents the Julian equivalent of the integer specified as argument-1. The returned value is an integer of the form YYYYDDD where YYYY represents a year in the Gregorian calendar and DDD represents the day of that year.

## FACTORIAL

The FACTORIAL function returns an integer that is the factorial of the argument specified.

The function type is integer.

**Format**




**argument-1**
Must be an integer greater than or equal to zero and less than or equal to 28.

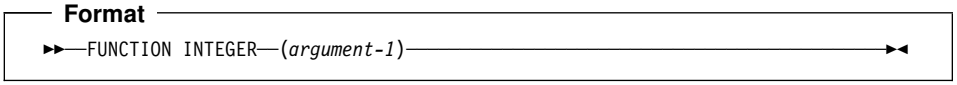
If the value of argument-1 is zero, the value 1 is returned; otherwise, its factorial is returned.

## INTEGER

The INTEGER function returns the greatest integer value that is less than or equal to the argument specified.

The function type is integer.

**Format**




**argument-1**
Must be class numeric.

The returned value is the greatest integer less than or equal to the value of argument-1. For example,

```
FUNCTION INTEGER (2.5)
```

will return a value of 2; and

```
FUNCTION INTEGER (-2.5)
```

will return a value of -3.

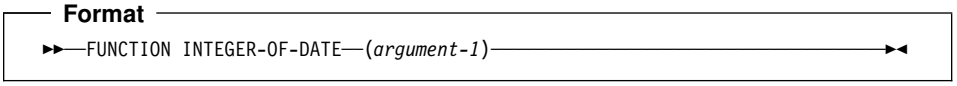
## INTEGER-OF-DATE

The INTEGER-OF-DATE function converts a date in the Gregorian calendar from standard date form (YYYYMMDD) to integer date form.

The function type is integer.

The function result is a seven-digit integer with a range from 1 to 3,067,671.

**Format**




**argument-1**

Must be an integer of the form YYYYMMDD, whose value is obtained from the calculation (YYYY * 10,000) + (MM * 100) + DD.

- YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600, but not greater than 9999.
- MM represents a month and must be a positive integer less than 13.
- DD represents a day and must be a positive integer less than 32, provided that it is valid for the specified month and year combination.

The returned value is an integer that is the number of days the date represented by argument-1, succeeds December 31, 1600 in the Gregorian calendar.

x MVS/VM Under MVS and VM, the INTDATE installation option affects the starting date
x for the integer date functions. For details, see the *IBM COBOL for MVS & VM Pro-*
x *gramming Guide*.

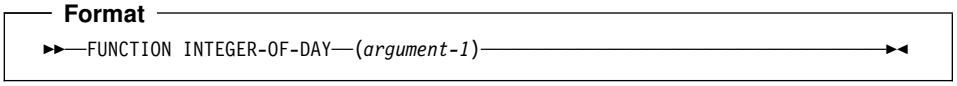
## INTEGER-OF-DAY

The INTEGER-OF-DAY function converts a date in the Gregorian calendar from Julian date form (YYYYDDD) to integer date form.

The function type is integer.

The function result is a seven-digit integer.

**Format**




**argument-1**

Must be an integer of the form YYYYDDD whose value is obtained from the calculation (YYYY * 1000) + DDD.

- YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600, but not greater than 9999.
- DDD represents the day of the year. It must be a positive integer less than 367, provided that it is valid for the year specified.

x MVS/VM Under MVS and VM, the INTDATE installation option affects the starting
x date for the integer date functions. For details, see the *IBM COBOL for MVS &*
x *VM Programming Guide*.

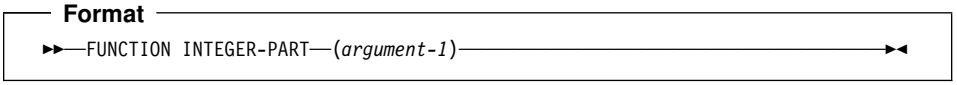
The returned value is an integer that is the number of days the date represented by argument-1, succeeds December 31, 1600 in the Gregorian calendar.

## INTEGER-PART

The INTEGER-PART function returns an integer that is the integer portion of the argument specified.

The function type is integer.

**Format**




**argument-1**
Must be class numeric.

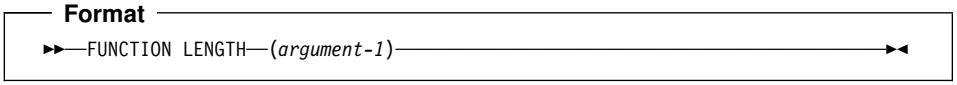
If the value of argument-1 is zero, the returned value is zero. If the value of argument-1 is positive, the returned value is the greatest integer less than or equal to the value of argument-1. If the value of argument-1 is negative, the returned value is the least integer greater than or equal to the value of argument-1.

## LENGTH

The LENGTH function returns an integer equal to the length of the argument in bytes. The function type is integer.

The function result is a nine-digit integer.

**Format**




**argument-1**

Can be a nonnumeric literal or a data item of any class or category (except DBCS).

If argument-1, or any data item subordinate to argument-1, is described with the DEPENDING phrase of the OCCURS clause, the contents of the data item referenced by the data-name specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated.

A data item described with USAGE IS POINTER or USAGE IS PROCEDURE-POINTER can be used as argument-1 to the LENGTH function.

The ADDRESS OF special register can be used as argument-1 to the LENGTH function.

If the ADDRESS OF special register or LENGTH OF special register is used as argument-1 to the LENGTH function, the result will always be 4, independent of the ADDRESS OF or LENGTH OF object.

If argument-1 is a nonnumeric literal, an elementary data item, or a group data item that does not contain a variable occurrence data item, the value returned is an integer equal to the length of argument-1 in character positions.

If argument-1 is a group data item containing a variable occurrence data item, the returned value is an integer determined by evaluation of the data item specified in the DEPENDING phrase of the OCCURS clause for that variable occurrence data item. This evaluation is accomplished according to the rules in the OCCURS clause regarding the data item as a sending data item. For more information, see the discussions of the OCCURS clause and USAGE clause.

If argument-1 is a null-terminated nonnumeric literal, then the returned value is the number of characters in the literal excluding the null character that is added at the end of the literal.

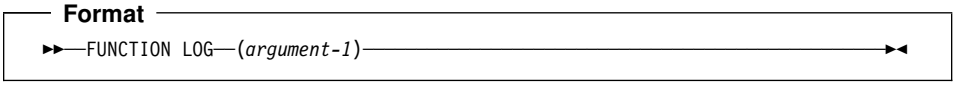
The returned value includes implicit FILLER characters, if any.

## LOG

The LOG function returns a numeric value that approximates the logarithm to the base ***e*** (natural log) of the argument specified.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric. The value of argument-1 must be greater than zero.

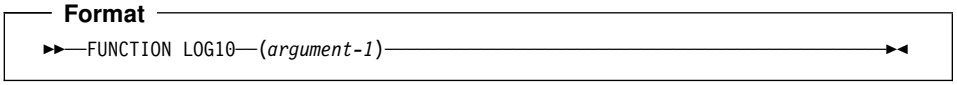
The returned value is the approximation of the logarithm to the base ***e*** of argument-1.

## LOG10

The LOG10 function returns a numeric value that approximates the logarithm to the base 10 of the argument specified.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric. The value of argument-1 must be greater than zero.

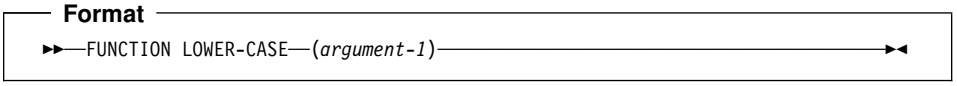
The returned value is the approximation of the logarithm to the base 10 of argument-1.

## LOWER-CASE

The LOWER-CASE function returns a character string that is the same length as the argument specified with each uppercase letter replaced by the corresponding lowercase letter.

The function type is alphanumeric.

**Format**




**argument-1**
Must be class alphabetic or alphanumeric and must be at least one character in length.

The same character string as argument-1 is returned, except that each uppercase letter is replaced by the corresponding lowercase letter.

The character string returned has the same length as argument-1.

## MAX

The MAX function returns the content of the argument that contains the maximum value.

The function type depends on the argument types, as follows:

| Argument Type | Function Type |
|---|---|
| Alphabetic | Alphanumeric |
| Alphanumeric | Alphanumeric |
| All arguments integer | Integer |
| Numeric (some arguments can be integer) | Numeric |

**Format**

**argument-1**
Must be class numeric, alphanumeric, or alphabetic.

x If more than one argument-1 is specified, the combination of alphabetic and alphanu-
x meric arguments is allowed.

The returned value is the content of argument-1 having the greatest value. The comparisons used to determine the greatest value are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

If more than one argument has the same greatest value, the leftmost argument having that value is returned.

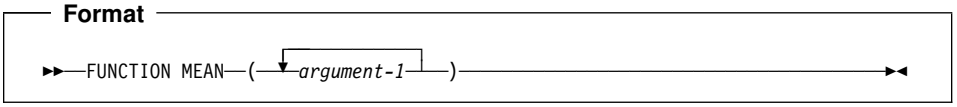
If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected argument.

## MEAN

The MEAN function returns a numeric value that approximates the arithmetic average of its arguments.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric.

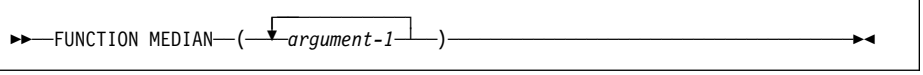
The returned value is the arithmetic mean of the argument-1 series. The returned value is defined as the sum of the argument-1 series divided by the number of occurrences referenced by argument-1.

## MEDIAN

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 function type is numeric.

**Format**




**argument-1**
Must be class numeric.

The returned value is the content of argument-1 having the middle value in the list formed by arranging all argument-1 values in sorted order.

If the number of occurrences referenced by 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. If the number of occurrences referenced by argument-1 is even, the returned value is the arithmetic mean of the values referenced by the two middle occurrences.

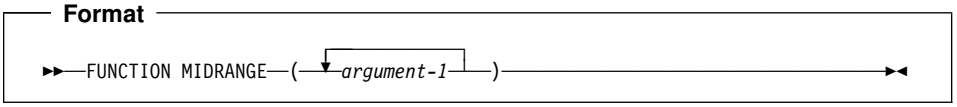
The comparisons used to arrange the argument values in sorted order are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

## MIDRANGE

The MIDRANGE function returns a numeric value that approximates the arithmetic average of the values of the minimum argument and the maximum argument.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric.

The returned value is the arithmetic mean of the value of the greatest argument-1 and the value of the least argument-1. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

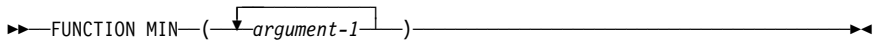
## MIN

The MIN function returns the content of the argument that contains the minimum value.

The function type depends on the argument types, as follows:

| Argument Type | Function Type |
|---|---|
| Alphabetic | Alphanumeric |
| Alphanumeric | Alphanumeric |
| All arguments integer | Integer |
| Numeric (some arguments can be integer) | Numeric |

**Format**




**argument-1**
Must be class numeric, alphanumeric, or alphabetic.

x If more than one argument-1 is specified, the combination of alphabetic and alphanu-
x meric arguments is allowed.

The returned value is the content of argument-1 having the least value. The comparisons used to determine the least value are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

If more than one argument-1 has the same least value, the leftmost argument-1 having that value is returned.

If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected argument-1.

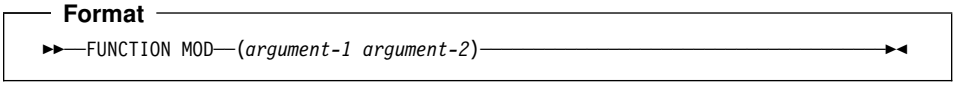
## MOD

The MOD function returns an integer value that is argument-1 modulo argument-2.

The function type is integer.

The function result is an integer with as many digits as the shorter of argument-1 and argument-2.

**Format**




**argument-1**
Must be an integer.

**argument-2**
Must be an integer. Must not be zero.

The returned value is argument-1 modulo argument-2. The returned value is defined as:

argument-1 - (argument-2 * FUNCTION INTEGER (argument-1 / argument-2))

The following illustrates the expected results for some values of argument-1 and argument-2.

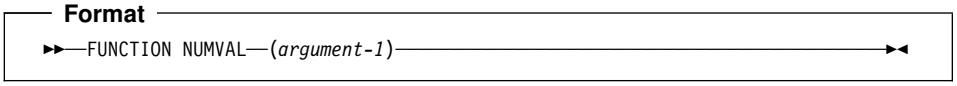
| Argument-1 | Argument-2 | Return |
|---|---|---|
| 11 | 5 | 1 |
| -11 | 5 | 4 |
| 11 | -5 | -4 |
| -11 | -5 | -1 |

## NUMVAL

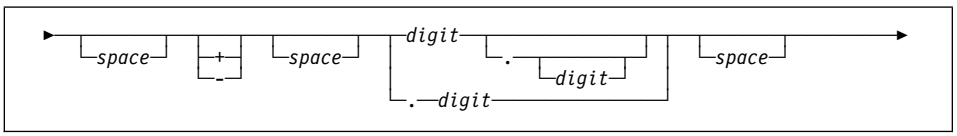
The NUMVAL function returns the numeric value represented by the alphanumeric character string specified in an argument. The function strips away any leading or trailing blanks in the string, producing a numeric value that can be used in an arithmetic expression.

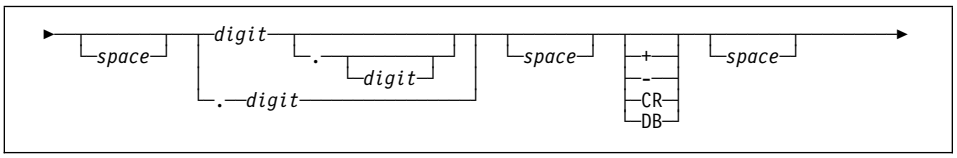
The function type is numeric.

**Format**




**argument-1**
must be a nonnumeric literal or an alphanumeric data item whose content has the following formats:




**space**
A string of one or more spaces.

**digit**
A string of one or more digits. The totals number of digits must not exceed 18.

If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma must be used in argument-1 rather than a decimal point.

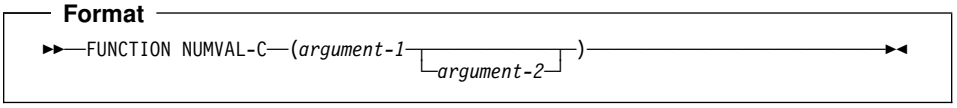
The returned value is an approximation of the numeric value represented by argument-1.

## NUMVAL-C

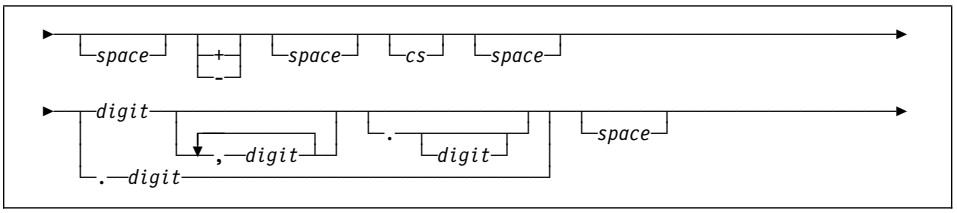
The NUMVAL-C function returns the numeric value represented by the alphanumeric character string specified as argument-1. Any optional currency sign specified by argument-2 and any optional commas preceding the decimal point are stripped away, producing a numeric value that can be used in an arithmetic expression.

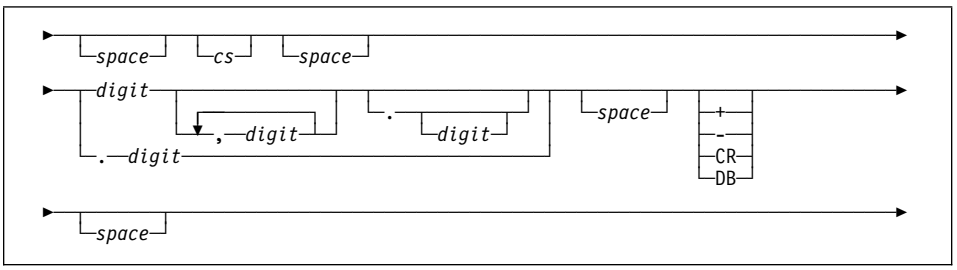
The function type is numeric.

**Format**




**argument-1**
Must be a nonnumeric literal or an alphanumeric data item whose content has the following formats:




**space**
A string of one or more spaces.

**cs** The string of one or more characters specified by argument-2. At most, one copy of the characters specified by cs can occur in argument-1.

**digit**
A string of one or more digits. The total number of digits must not exceed 18.

If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the functions of the comma and decimal point in argument-1 are reversed.

**argument-2**

If specified, must be a nonnumeric literal or alphanumeric data item, subject to the following rules:

- argument-2 must not contain any of the digits 0 through 9, any leading or trailing spaces, or any of the special characters + - . ,
- argument-2 can be of any length valid for an elementary or group data item, including zero
- Matching of argument-2 is case-sensitive. For example, if you specify argument-2 as 'Dm', it will not match 'DM', 'dm' or 'dM'.

If argument-2 is not specified, the character used for cs is the currency symbol specified for the program.

The returned value is an approximation of the numeric value represented by argument-1.

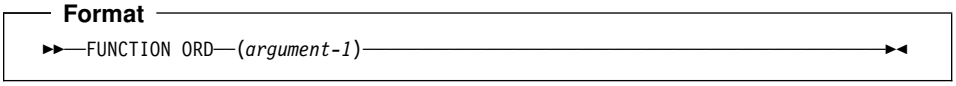
## ORD

The ORD function returns an integer value that is the ordinal position of its argument in the collating sequence for the program. The lowest ordinal position is 1.

The function type is integer.

The function result is a three-digit integer.

**Format**




**argument-1**
Must be one character in length and must be class alphabetic or alphanumeric.

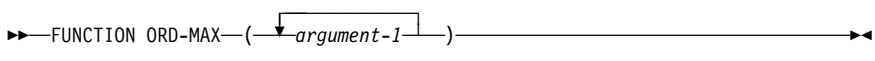
The returned value is the ordinal position of argument-1 in the collating sequence for the program; it ranges from 1 to 256 depending on the collating sequence.

## ORD-MAX

The ORD-MAX function returns a value that is the ordinal number position, in the argument list, of the argument that contains the maximum value.

The function type is integer.

**Format**




**argument-1**
Must be class numeric, alphanumeric, or alphabetic.

x If more than one argument-1 is specified, the combination of alphabetic and alphanu-
x meric arguments is allowed.

The returned value is the ordinal number that corresponds to the position of argument-1 having the greatest value in the argument-1 series.

The comparisons used to determine the greatest valued argument-1 are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

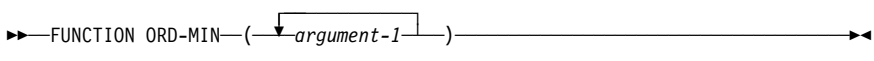
If more than one argument-1 has the same greatest value, the number returned corresponds to the position of the leftmost argument-1 having that value.

## ORD-MIN

The ORD-MIN function returns a value that is the ordinal number of the argument that contains the minimum value.

The function type is integer.

**Format**




**argument-1**
Must be class numeric, alphanumeric, or alphabetic.

If more than one argument-1 is specified, the combination of alphabetic and alphanumeric arguments is allowed.

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.

The comparisons used to determine the least valued argument-1 are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

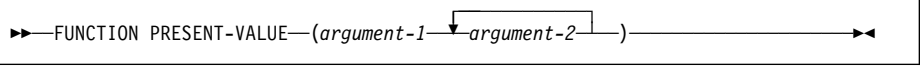
If more than one argument-1 has the same least value, the number returned corresponds to the position of the leftmost argument-1 having that value.

## PRESENT-VALUE

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 function type is numeric.

**Format**




**argument-1**
Must be class numeric. Must be greater than -1.

**argument-2**
Must be class numeric.

The returned value is an approximation of the summation of a series of calculations with each term in the following form:

argument-2 / (1 + argument-1) ** n

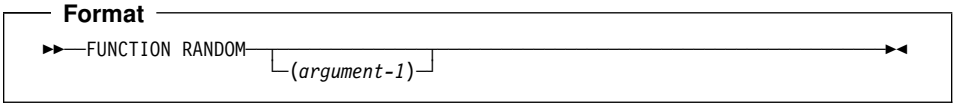
There is one term for each occurrence of argument-2. The exponent, n, is incremented from one by one for each term in the series.

## RANDOM

The RANDOM function returns a numeric value that is a pseudo-random number from a rectangular distribution.

The function type is numeric.

**Format**




**argument-1**

If argument-1 is specified, it must be zero or a positive integer, up to and including (10**18)-1 which is the maximum value that can be specified in a PIC9(18) fixed item; however, only those in the range from zero up to and including 2,147,483,645 will yield a distinct sequence of pseudo-random numbers.

If a subsequent reference specifies argument-1, a new sequence of pseudo-random numbers is started.

If the first reference to this function in the run unit does not specify argument-1, the seed value used will be zero.

In each case, subsequent references without specifying argument-1 return the next number in the current sequence.

The returned value is exclusively between zero and one.

For a given seed value, the sequence of pseudo-random numbers will always be the same.

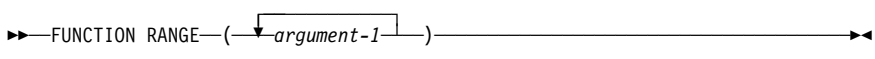
## RANGE

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument.

The function type depends on the argument types, as follows:

| Argument Type | Function Type |
|---|---|
| All arguments integer | Integer |
| Numeric (some arguments can be integer) | Numeric |

**Format**




**argument-1**
Must be class numeric.

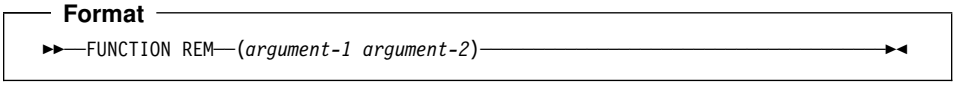
The returned value is equal to argument-1 with the greatest value minus the argument-1 with the least value. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

## REM

The REM function returns a numeric value that is the remainder of argument-1 divided by argument-2.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric

**argument-2**
Must be class numeric. Must not be zero.

The returned value is the remainder of argument-1 divided by argument-2. It is defined as the expression:

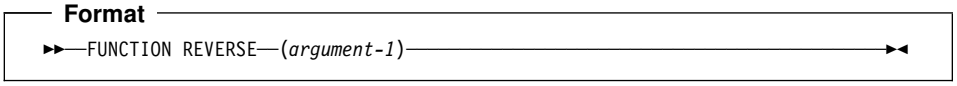
argument-1 - (argument-2 * FUNCTION INTEGER-PART (argument-1/argument-2))

## REVERSE

The REVERSE function returns a character string of exactly the same length of the argument, whose characters are exactly the same as those specified in the argument, except that they are in reverse order.

The function type is alphanumeric.

**Format**




**argument-1**
Must be class alphabetic or alphanumeric and must be at least one character in length.

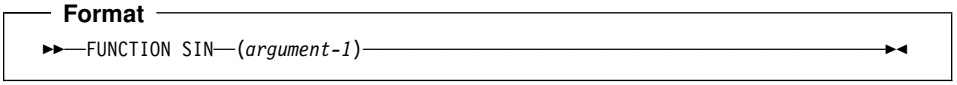
If argument-1 is a character string of length n, the returned value is a character string of length n such that, for 1 <= j <= n, the character in position j of the returned value is the character from position n-j+1 of argument-1.

## SIN

The SIN function returns a numeric value that approximates the sine of the angle or arc specified by the argument in radians.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric.

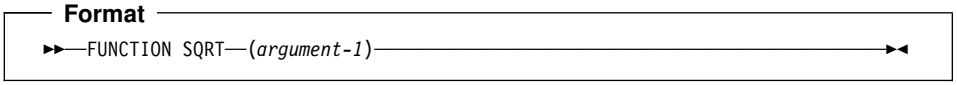
The returned value is the approximation of the sine of argument-1 and is greater than or equal to -1 and less than or equal to +1.

## SQRT

The SQRT function returns a numeric value that approximates the square root of the argument specified.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric. The value of argument-1 must be zero or positive.

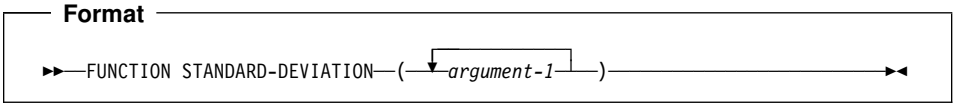
The returned value is the absolute value of the approximation of the square root of argument-1.

## STANDARD-DEVIATION

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric.

The returned value is the approximation of the standard deviation of the argument-1 series. The returned value is calculated as follows:

1. The difference between each argument-1 and the arithmetic mean of the argument-1 series is calculated and squared.
2. The values obtained are then added together. This quantity is divided by the number of values in the argument-1 series.
3. The square root of the quotient obtained is then calculated. The returned value is the absolute value of this square root.

If the argument-1 series consists of only one value, or if the argument-1 series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.

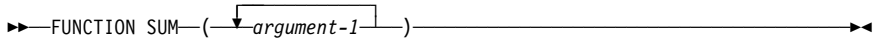
## SUM

The SUM function returns a value that is the sum of the arguments.

The function type depends on the argument types, as follows:

| Argument Type | Function Type |
|---|---|
| All arguments integer | Integer |
| Numeric (some arguments can be integer) | Numeric |

**Format**




**argument-1**
Must be class numeric.

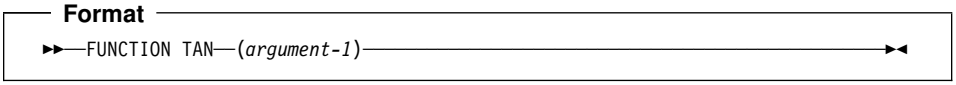
The returned value is the sum of the arguments. If the argument-1 series are all integers, the value returned is an integer. If the argument-1 series are not all integers, a numeric value is returned.

## TAN

The TAN function returns a numeric value that approximates the tangent of the angle or arc that is specified by the argument in radians.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric.

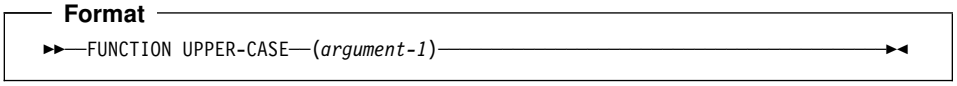
The returned value is the approximation of the tangent of argument-1.

## UPPER-CASE

The UPPER-CASE function returns a character string that is the same length as the argument specified, with each lowercase letter replaced by the corresponding uppercase letter.

The function type is alphanumeric.

**Format**




**argument-1**
Must be class alphabetic or alphanumeric and must be at least one character in length.

The same character string as argument-1 is returned, except that each lowercase letter is replaced by the corresponding uppercase letter.

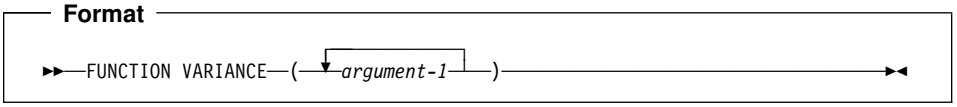
The character string returned has the same length as argument-1.

## VARIANCE

The VARIANCE function returns a numeric value that approximates the variance of its arguments.

The function type is numeric.

**Format**




**argument-1**
Must be class numeric.

The returned value is the approximation of the variance of the argument-1 series.

The returned value is defined as the square of the standard deviation of the argument-1 series. This value is calculated as follows:

1. The difference between each argument-1 value and the arithmetic mean of the argument-1 series is calculated and squared.
2. The values obtained are then added together. This quantity is divided by the number of values in the argument series.

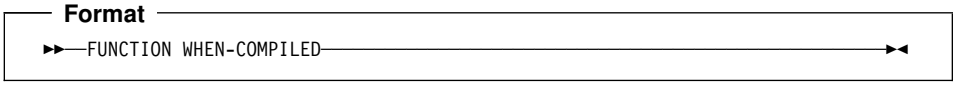
If the argument-1 series consists of only one value, or if the argument-1 series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.

## WHEN-COMPILED

The WHEN-COMPILED function returns the date and time the program was compiled as provided by the system on which the program was compiled.

The function type is alphanumeric.

**Format**




Reading from left to right, the 21 character positions in the value returned can be interpreted as follows:

| 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 digits of the seconds past the minute, in the range 00 through 59. |
| **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 '-' or the character '+'. The character '-' is returned if the local time indicated in the previous character positions is behind Greenwich Mean Time. The character '+' is returned if the local time indicated is the same as or ahead of Greenwich Mean 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 reported time is behind Greenwich Mean Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich Mean 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 reported time is ahead of or behind Greenwich Mean Time, depending on whether character position 17 is '+' or '-', respectively. If character position 17 is '0', the value 00 is returned. |

## WHEN-COMPILED

The returned value is the date and time of compilation of the source program that contains this function. If the program is a contained program, the returned value is the compilation date and time associated with the separately compiled program in which it is contained.

## WHEN-COMPILED

# Part 8. Compiler-Directing Statements


**Compiler-Directing Statement** . . . . . 482
x BASIS Statement . . . . . 482
x CBL (PROCESS) Statement . . . . . 483
x *CONTROL (*CBL) Statement . . . . . 484
COPY Statement . . . . . 486
x DELETE Statement . . . . . 493
x EJECT Statement . . . . . 495
ENTER Statement . . . . . 496
x INSERT Statement . . . . . 497
x READY or RESET TRACE Statement . . . . . 498
REPLACE Statement . . . . . 500
x SERVICE LABEL Statement . . . . . 504
x SERVICE RELOAD Statement . . . . . 505
x SKIP1/2/3 Statements . . . . . 506
x TITLE Statement . . . . . 507
USE Statement . . . . . 508

**Compiler Directives** . . . . . 514
x CALLINTERFACE . . . . . 514


© Copyright IBM Corp. 1991, 1996

# Compiler-Directing Statement

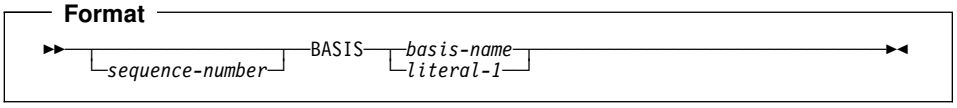
A **Compiler-Directing Statement** is a statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

## BASIS Statement

The BASIS statement is an extended source program library statement. It provides a complete COBOL program as the source for a compilation.

A complete program can be stored as an entry in a user's library and can be used as the source for a compilation. Compiler input is a BASIS statement, optionally followed by any number of INSERT and/or DELETE statements.

**Format**




**sequence-number**
Can optionally appear in columns 1 through 6, followed by a space. The content of this field is ignored.

**BASIS**
Can appear anywhere in columns 1 through 72, followed by basis-name. There must be no other text in the statement.

**basis-name, literal-1**
It is the name by which the library entry is known to the system environment.

For rules of formation and processing rules, see the description under literal-1 and text-name-1 of the “COPY Statement” on page 486.

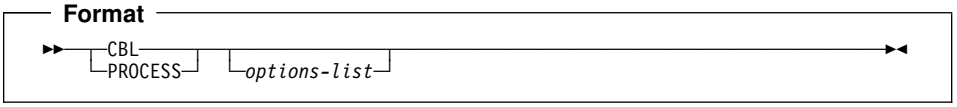
The source file remains unchanged after execution of the BASIS statement.

**Note:** If INSERT or DELETE statements are used to modify the COBOL source program provided by a BASIS statement, the sequence field of the COBOL source program must contain numeric sequence numbers in ascending order.

## CBL (PROCESS) Statement

With the CBL (PROCESS) statement, you can specify compiler options to be used in the compilation of the program. The CBL (PROCESS) statement is placed before the Identification Division header of an outermost program.

**Format**




**options-list**

A series of one or more compiler options, each one separated by a comma or a space.

For more information on compiler options, see the *IBM COBOL Programming Guide* for your platform.

The CBL (PROCESS) statement can be preceded by a sequence number in columns 1 through 6. The first character of the sequence number must be numeric, and CBL or PROCESS can begin in column 8 or after; if a sequence number is not specified, CBL or PROCESS can begin in column 1 or after.

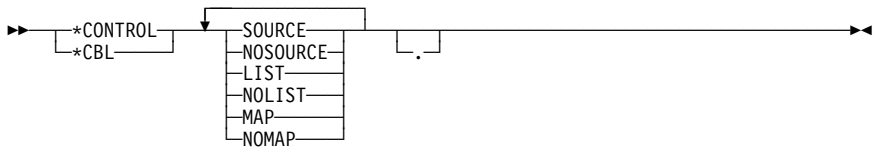
The CBL (PROCESS) statement must end before or at column 72, and options cannot be continued across multiple CBL (PROCESS) statements. However, you can use more than one CBL (PROCESS) statement. If you use multiple CBL (PROCESS) statements, they must follow one another with no intervening statements of any other type.

The CBL (PROCESS) statement must be placed before any comment lines or other compiler-directing statements.

## *CONTROL (*CBL) Statement

With the *CONTROL (or *CBL) statement, you can selectively display or suppress the listing of source code, object code, and storage maps throughout the source program.

**Format**




(For a complete discussion of the output produced by these options, see the *IBM COBOL Programming Guide* for your platform.)

The *CONTROL and *CBL statements are synonymous. Whenever *CONTROL is used, *CBL is accepted as well.

The characters *CONTROL or *CBL can start in any column beginning with column 7, followed by at least one space or comma and one or more option key words. The option key words must be separated by one or more spaces or commas. This statement must be the only statement on the line, and continuation is not allowed. The statement can be terminated with a period.

The *CONTROL and *CBL statements must be embedded in a program source. For example, in the case of batch jobs the *CONTROL and *CBL statements must be placed between the PROCESS (CBL) statement and the end of the program (or END PROGRAM header, if specified).

The source line containing the *CONTROL (*CBL) statement will not appear in the source listing.

If an option is defined at installation as a fixed option, this fixed option takes precedence over all of the following:

- PARM (if available)
- CBL statement
- *CONTROL (*CBL) statement

The requested options are handled in the following manner:

1. If an option or its negation appears more than once in a *CONTROL statement, the last occurrence of the option word is used.
2. If the CORRESPONDING option has been requested as a parameter to the compiler, then a *CONTROL statement with the negation of the option word must precede the portions of the source program for which listing output is to be inhibited. Listing output then resumes when a *CONTROL statement with the affirmative option word is encountered.

3. If the negation of the CORRESPONDING option has been requested as a parameter to the compiler, then that listing is **always** inhibited.
4. The *CONTROL statement is in effect only within the source program in which it is written, including any contained programs. It does not remain in effect across batch compiles of two or more COBOL source programs.

## Source Code Listing

Listing of the input source program lines is controlled by any of the following statements:

```
*CONTROL SOURCE             [*CBL SOURCE]
*CONTROL NOSOURCE           [*CBL NOSOURCE]
```

If a *CONTROL NOSOURCE statement is encountered and SOURCE has been requested as a compilation option, printing of the source listing is suppressed from this point on. An informational (I-level) message is issued stating that `PRINTING OF THE SOURCE HAS BEEN SUPPRESSED`.

## Object Code Listing

Listing of generated object code is controlled by any of the following statements occurring in the Procedure Division:

```
*CONTROL LIST               [*CBL LIST]
*CONTROL NOLIST             [*CBL NOLIST]
```

If a *CONTROL NOLIST statement is encountered, and LIST has been requested as a compilation option, listing of generated object code is suppressed from this point on.

## Storage Map Listing

Listing of storage map entries is controlled by any of the following statements occurring in the Data Division:

```
*CONTROL MAP                [*CBL MAP]
*CONTROL NOMAP              [*CBL NOMAP]
```

If a *CONTROL NOMAP statement is encountered, and MAP has been requested as a compilation option, listing of storage map entries is suppressed from this point on.

For example, either of the following sets of statements produces a storage map listing in which A and B will not appear:

```
*CONTROL NOMAP           *CBL NOMAP
    01  A                    01  A
    02  B                    02  B
*CONTROL MAP             *CBL MAP
```

## COPY Statement

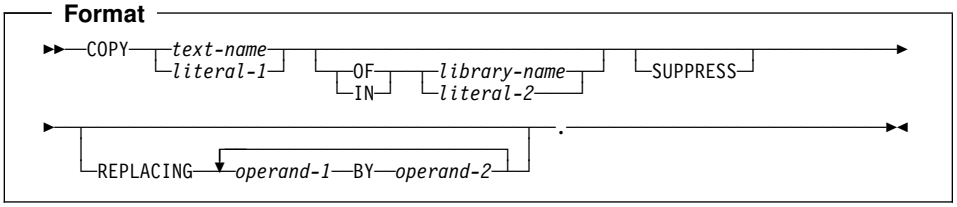
The COPY statement is a library statement that places prewritten text in a COBOL program.

Prewritten source program entries can be included in a source program at compile time. Thus, an installation can use standard file descriptions, record descriptions, or procedures without recoding them. These entries and procedures can then be saved in user-created libraries; they can then be included in the source program by means of the COPY statement.

Compilation of the source program containing COPY statements is logically equivalent to processing all COPY statements before processing the resulting source program.

The effect of processing a COPY statement is that the library text associated with text-name is copied into the source program, logically replacing the entire COPY statement, beginning with the word COPY and ending with the period, inclusive. When the REPLACING phrase is not specified, the library text is copied unchanged.

**Format**




**text-name, library-name**

Text-name identifies the name of the copy text. Library-name identifies where the copy text exists.

Must follow the normal rules of formation for a user-defined word:

- The word can be from 1-30 characters in length
- Can contain characters A-Z, a-z, 0-9, hyphen
- At least one characters must be alphabetic
- The hyphen can't be the first or last character

**literal-1, literal-2**

Literal-1 identifies the name of the copy text. Literal-2 identifies where the copy text exists.

Must conform to the following rules of formation:

- The literal can be from 1-160 characters in length
- For MVS and VM, can contain characters A-Z, a-z, 0-9, hyphen, @, #, and $
- For MVS and VM, the leading character must be alphabetic

The uniqueness of text-name and library-name is determined after the formation and conversion rules for a system-dependent name have been applied.

x As an IBM extension, a user-defined word can be the same as a text-name or a library-
x name.

x As an IBM extension, if more than one COBOL library is available during compilation,
x text-name need not be qualified. If text-name is not qualified, a library-name of SYSLIB
x is assumed.

| For information on processing rules, see the *IBM COBOL Programming Guide* for your
| platform.

**operand-1, operand-2**
Can be either pseudo-text, an identifier, a function-identifier, a literal, or a COBOL word (except COPY).

Each COPY statement must be preceded by a space and ended with a separator period.

A COPY statement can appear in the source program anywhere a character string or a
x separator can appear. As an IBM extension, COPY statements can be nested.
x However, nested COPY statements cannot contain the REPLACING phrase, and a
x COPY statement with the REPLACING phrase cannot contain nested COPY state-
x ments.

x A COPY statement cannot cause recursion. That is, a COPY member can be named
x only once in a set of nested COPY statements until the end-of-file for that COPY
x member is reached. For example, assume that the source program contains the state-
x ment: `COPY X.` and library-text X contains the statement: `COPY Y.`.

x In this case, the library-text Y must not have a `COPY X` or a `COPY Y` statement.

Debugging lines are permitted within library text and pseudo-text. Text words within a debugging line participate in the matching rules as if the D did not appear in the indicator area. A debugging line is specified within pseudo-text if the debugging line begins in the source program after the opening pseudo-text delimiter but before the matching closing pseudo-text delimiter.

If additional lines are introduced into the source program as a result of a COPY statement, each text word introduced appears on a debugging line if the COPY statement begins on a debugging line or if the text word being introduced appears on a debugging line in Library text. When a text word specified in the BY phrase is introduced, it appears on a debugging line if the first library text word being replaced is specified on a debugging line.

When a COPY statement is specified on a debugging line, the copied text is treated as though it appeared on a debugging line, except that comment lines in the text appear as comment lines in the resulting source program.

If the word COPY appears in a comment-entry, or in the place where a comment-entry can appear, it is considered part of the comment-entry.

## COPY Statement

After all COPY and REPLACE statements have been processed, a debugging line will be considered to have all the characteristics of a comment line, if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

Comment lines or blank lines can occur in library text. Comment lines or blank lines appearing in library text are copied into the resultant source program unchanged with the following exception: a comment line or blank line in library text is not copied if that comment line or blank line appears within the sequence of text words that match operand-1 (refer to "Replacement and Comparison Rules" on page 489).

x Lines containing *CONTROL (*CBL), EJECT, SKIP1/2/3, or TITLE statements can
x occur in library text. Such lines are treated as comment lines during COPY statement
x processing.

The syntactic correctness of the entire COBOL source program cannot be determined until all COPY and REPLACE statements have been completely processed, because the syntactic correctness of the library text cannot be independently determined.

Library text copied from the library is placed into the same area of the resultant program as it is in the library. Library text must conform to the rules for standard COBOL format.

**Note:** Characters outside the standard COBOL character set must not appear in library text or pseudo-text, other than as part of nonnumeric literals, comment lines, or comment-entries.

x DBCS words, DBCS literals, and EUC words are allowed in library text and pseudo-
x text.

## SUPPRESS Phrase

x The SUPPRESS phrase specifies that the library text is not to be printed on the source
x program listing.

## REPLACING Phrase

In the discussion that follows, each **operand** can consist of one of the following:

- Pseudo-text
- An identifier
- A literal
- A COBOL word (except COPY)
- Function identifier

When the REPLACING phrase is specified, the library text is copied, and each properly matched occurrence of operand-1 within the library text is replaced by the associated operand-2.

**pseudo-text**
A sequence of character-strings and/or separators bounded by, but not including, pseudo-text-1 delimiters (==). Both characters of each pseudo-text-1 delimiter

must appear on one line; however, character-strings within pseudo-text-1 can be continued.

Any individual character-string within pseudo-text-1 can be up to 322 characters long. Keep in mind that a character-string must be delimited by separators. For more information, see "Characters" on page 4.

Pseudo-text-1 refers to pseudo-text when used for operand-1, and pseudo-text-2 refers to pseudo-text when used for operand-2.

Pseudo-text-1 cannot be null, nor can it consist solely of the space character, separator comma, separator semicolon, and/or of comment lines. Beginning and ending blanks are not included in the text comparison process. Embedded blanks are used in the text comparison process to indicate multiple text words.

Pseudo-text must not contain the word `COPY`.

x Pseudo-text-1 can consist solely of the separator comma or separator semicolon.

Pseudo-text-2 can be null; it can consist solely of space characters and/or comment lines. Each text word in pseudo-text-2 that is to be copied into the program is placed in the same area of the resultant program as the area in which it appears in pseudo-text-2.

x Pseudo-text can contain DBCS or EUC user-defined words, but the characters
x cannot be continued across lines.

**identifier**
Can be defined in any Data Division section.

**literal**
Can be numeric or nonnumeric.

x Can be a DBCS literal.

**word**
Can be any single COBOL word (except COPY).

x You can include the non-separator COBOL characters (for example, +, *, /, $, <, >,
x and =) as part of a COBOL word when used as REPLACING operands. In addi-
x tion, the hyphen character can be at the beginning or end of the word.

For purposes of matching, each identifier-1, literal-1, or word-1 is treated, respectively, as pseudo-text containing only identifier-1, literal-1, or word-1.

x Either operand, or both, can be a DBCS or EUC name or DBCS literal.

## Replacement and Comparison Rules

1. Arithmetic and logical operators are considered text words and can be replaced only through the pseudo-text option.
2. When a figurative constant is operand-1, it will match only if it appears exactly as it is specified. For example, if ALL "AB" is specified in the library text, then "ABAB" is not considered a match; only ALL "AB" is considered a match.

3. When replacing a PICTURE character-string, the pseudo-text option should be used; to avoid ambiguities, pseudo-text-1 should specify the entire PICTURE clause, including the key word PICTURE or PIC.
4. Any separator comma, semicolon, and/or space preceding the leftmost word in the library text is copied into the source program. Beginning with the leftmost library text word and the first operand-1 specified in the REPLACING option, the entire REPLACING operand that precedes the key word BY is compared to an equivalent number of contiguous library text words.
5. Operand-1 matches the library text if, and only if, the ordered sequence of text words in operand-1 is equal, character for character, to the ordered sequence of library words. For matching purposes, each occurrence of a comma or semicolon separator and each sequence of one or more space separators is considered to be a single space. However, when operand-1 consists solely of a separator comma or semicolon, it participates in the match as a text-word (in this case, the space following the comma or semicolon separator can be omitted).
6. If no match occurs, the comparison is repeated with each successive operand-1, if specified, until either a match is found or there are no further REPLACING operands.
7. Whenever a match occurs between operand-1 and the library text, the associated operand-2 is copied into the source program.
8. The COPY statement with REPLACING phrase can be used to replace parts of words. By inserting a dummy operand delimited by colons into the program text, the compiler will replace the dummy operand with the desired text. Example 3 shows how this is used with the dummy operand `:TAG:`.

   **Note:** The colons serve as separators and make TAG a stand-alone operand.
9. When all operands have been compared and no match is found, the leftmost library text word is copied into the source program.
10. The next successive uncopied library text word is then considered to be the leftmost text word, and the comparison process is repeated, beginning with the first operand-1. The process continues until the rightmost library text word has been compared.
11. Comment lines or blank lines occurring in the library text and in pseudo-text-1 are ignored for purposes of matching; and the sequence of text words in the library text and in pseudo-text-1 is determined by the rules for reference format. Comment lines or blank lines appearing in pseudo-text-2 are copied into the resultant program unchanged whenever pseudo-text-2 is placed into the source program as a result of text replacement. Comment lines or blank lines appearing in library text are copied into the resultant source program unchanged with the following exception: a comment line or blank line in library text is not copied if that comment line or blank line appears within the sequence of text words that match pseudo-text-1.
12. Text words, after replacement, are placed in the source program according to standard COBOL format rules.

x 13. COPY REPLACING does not affect the EJECT, SKIP1/2/3, or TITLE compiler-
x directing statements. When text words are placed in the source program, addi-
x tional spaces are introduced only between text words where there already exists a
x space (including the assumed space between source lines).

Sequences of code (such as file and data descriptions, error and exception routines, etc.) that are common to a number of programs can be saved in a library, and then used in conjunction with the COPY statement. If naming conventions are established for such common code, then the REPLACING phrase need not be specified. If the names will change from one program to another, then the REPLACING phrase can be used to supply meaningful names for this program.

**Example 1**

In this example, the library text PAYLIB consists of the following Data Division entries:

```
01  A.
  02  B    PIC S99.
  02  C    PIC S9(5)V99.
  02  D    PIC S9999 OCCURS 1 TO 52 TIMES
      DEPENDING ON B OF A.
```

The programmer can use the COPY statement in the Data Division of a program as follows:

```
    COPY PAYLIB.
```

In this program, the library text is copied; the resulting text is treated as if it had been written as follows:

```
01  A.
  02  B    PIC S99.
  02  C    PIC S9(5)V99.
  02  D    PIC S9999 OCCURS 1 TO 52 TIMES
      DEPENDING ON B OF A.
```

**Example 2**

To change some (or all) of the names within the library text, the programmer can use the REPLACING phrase:

```
  COPY PAYLIB REPLACING  A BY PAYROLL
                         B BY PAY-CODE
                         C BY GROSS-PAY
                         D BY HOURS.
```

In this program, the library text is copied; the resulting text is treated as if it had been written as follows:

```
01  PAYROLL.
  02  PAY-CODE    PIC S99.
  02  GROSS-PAY    PIC S9(5)V99.
  02  HOURS    PIC S9999 OCCURS 1 TO 52 TIMES
      DEPENDING ON PAY-CODE OF PAYROLL.
```

The changes shown are made only for this program. The text, as it appears in the library, remains unchanged.

**Example 3**

If the following conventions are followed in library text, then parts of names (for example the prefix portion of data-names) can be changed with the REPLACING phrase.

In this example, the library text PAYLIB consists of the following Data Division entries:

```
01  :TAG:.
  02  :TAG:-WEEK          PIC S99.
  02  :TAG:-GROSS-PAY     PIC S9(5)V99.
  02  :TAG:-HOURS         PIC S999  OCCURS 1 TO 52 TIMES
      DEPENDING ON :TAG:-WEEK OF :TAG:.
```

The programmer can use the COPY statement in the Data Division of a program as follows:

```
COPY PAYLIB REPLACING ==:TAG:== BY ==Payroll==.
```

**Note:** It is important to notice in this example the required use of colons or parentheses as delimiters in the library text. Colons are recommended for clarity because parentheses can be used for a subscript, for instance in a table.

In this program, the library text is copied; the resulting text is treated as if it had been written as follows:

```
01  PAYROLL.
  02  PAYROLL-WEEK        PIC S99.
  02  PAYROLL-GROSS-PAY   PIC S9(5)V99.
  02  PAYROLL-HOURS       PIC S999  OCCURS 1 TO 52 TIMES
      DEPENDING ON PAYROLL-WEEK OF PAYROLL.
```

The changes shown are made only for this program. The text, as it appears in the library, remains unchanged.

**Example 4**

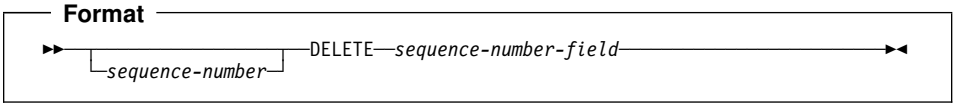
This example shows how to selectively replace level numbers without replacing the numbers in the PICTURE clause:

```
COPY xxx REPLACING ==(01)== BY ==(01)==
                   == 01 == BY == 05 ==.
```

## DELETE Statement

The DELETE statement is an extended source library statement. It removes COBOL statements from the source program included by a BASIS statement.

**Format**




**sequence-number**
Can optionally appear in columns 1 through 6, followed by a space. The content of this field is ignored.

**DELETE**
Can appear anywhere within columns 1 through 72. It must be followed by a space and the sequence-number-field. There must be no other text in the statement.

**sequence-number-field**
Each number must be equal to a sequence-number in the BASIS source program. This sequence-number is the 6-digit number the programmer assigns in columns 1 through 6 of the COBOL coding form. The numbers referenced in the sequence-number-fields of any INSERT or DELETE statements must always be specified in ascending numeric order.

The sequence-number-field must be any one of the following:

- A single number
- A series of single numbers
- A range of numbers (indicated by separating the two bounding numbers of the range by a hyphen)
- A series of ranges of numbers
- Any combination of one or more single numbers and one or more ranges of numbers

Each entry in the sequence-number-field must be separated from the preceding entry by a comma followed by a space. For example:

```
000250 DELETE  000010-000050, 000400, 000450
```

Source program statements can follow a DELETE statement. These source program statements are then inserted into the BASIS source program before the statement following the last statement deleted (that is, in the example above, before the next statement following deleted statement 000450).

If a DELETE statement is specified, beginning in column 12 or higher, and a valid sequence-number-field does not follow the key word DELETE, the compiler assumes that this DELETE statement is the COBOL DELETE statement.

## DELETE Statement

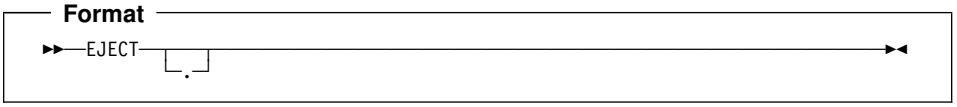
New source program statements following the DELETE statement can include DBCS or EUC data items.

**Note:** If INSERT or DELETE statements are used to modify the COBOL source program provided by a BASIS statement, the sequence field of the COBOL source program must contain numeric sequence-numbers in ascending order. The source file remains unchanged. Any INSERT or DELETE statements referring to these sequence-numbers must occur in ascending order.

## EJECT Statement

The EJECT statement specifies that the next source statement is to be printed at the top of the next page.

**Format**




The EJECT statement must be the only statement on the line. It can be written in either Area A or Area B, and can be terminated with a separator period.

The EJECT statement must be embedded in a program source. For example, in the case of batch jobs, the EJECT statement must be placed between the CBL (PROCESS) statement and the end of the program (or the END PROGRAM header, if specified).

The EJECT statement has no effect on the compilation of the source program itself.

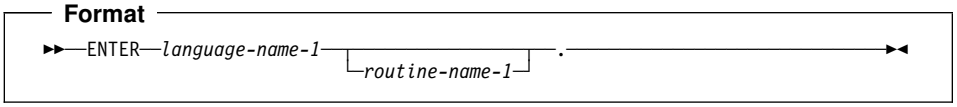
## ENTER Statement

The ENTER statement allows the use of more than one source language in the same source program.

With IBM COBOL for MVS & VM, IBM COBOL Set for AIX, and IBM VisualAge for COBOL for OS/2, only COBOL is allowed in the source program.

**Note:** The ENTER statement is syntax checked during compilation but has no effect on the execution of the program.

**Format**



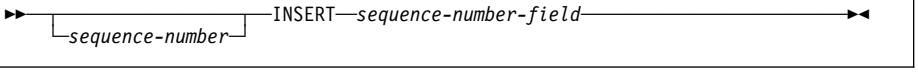

**language-name-1**
A system name that has no defined meaning. It must be either a correctly formed user-defined word or the word "COBOL". At least one character must be alphabetic.

**routine-name-1**
Must follow the rules for formation of a user-defined word. At least one character must be alphabetic.

## INSERT Statement

The INSERT statement is a library statement that adds COBOL statements to the source program included by a BASIS statement.

**Format**




**sequence-number**
Can optionally appear in columns 1 through 6, followed by a space. The content of this field is ignored.

**INSERT**
Can appear anywhere within columns 1 through 72, followed by a space and the sequence-number-field. There must be no other text in the statement.

**sequence-number-field**
A number which must be equal to a sequence-number in the BASIS source program. This sequence-number is the 6-digit number the programmer assigns in columns 1 through 6 of the COBOL coding form.

The numbers referenced in the sequence-number-fields of any INSERT or DELETE statements must always be specified in ascending numeric order.

The sequence-number-field must be a single number (for example, 000130). At least one new source program statement must follow the INSERT statement for insertion after the statement number specified by the sequence-number-field.

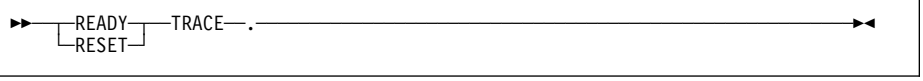
New source program statements following the INSERT statement can include DBCS or EUC data items.

**Note:** If INSERT or DELETE statements are used to modify the COBOL source program provided by a BASIS statement, the sequence field of the COBOL source program must contain numeric sequence-numbers in ascending order. The source file remains unchanged. Any INSERT or DELETE statements referring to these sequence-numbers must occur in ascending order.

## READY or RESET TRACE Statement

The READY or RESET TRACE statement can only appear in the Procedure Division, but has no effect on your program.

**Format**




You can reproduce the function of READY TRACE by using the USE FOR DEBUGGING declarative, DISPLAY statement, and DEBUG-ITEM special register. For example:

```
           .
           .
   ENVIRONMENT DIVISION.
     CONFIGURATION SECTION.
     SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.
           .
   DATA DIVISION.
           .
     WORKING-STORAGE SECTION.
     01 TRACE-SWITCH        PIC 9 VALUE 0.
        88  READY-TRACE           VALUE 1.
        88  RESET-TRACE           VALUE 0.
           .
   PROCEDURE DIVISION.
     DECLARATIVES.
     COBOL-II-DEBUG SECTION.
       USE FOR DEBUGGING ON ALL PROCEDURES.
     COBOL-II-DEBUG-PARA.
       IF READY-TRACE THEN
           DISPLAY DEBUG-NAME
       END-IF.
     END DECLARATIVES.
     MAIN-PROCESSING SECTION.
           .
     PARAGRAPH-3.
           .
       SET READY-TRACE TO TRUE.
     PARAGRAPH-4.
           .
     PARAGRAPH-6.
           .
       SET RESET-TRACE TO TRUE.
     PARAGRAPH-7.
```

where DEBUG-NAME is a field of the DEBUG-ITEM special register that displays the procedure-name causing execution of the debugging procedure. (In this example, the object program displays the names of procedures PARAGRAPH-4 through PARAGRAPH-6 as control reaches each procedure within the range.)

- At run time, you must specify the DEBUG run-time option to activate this debugging procedure. In this way, you have no need to recompile the program to activate or deactivate the debugging declarative.

## REPLACE Statement

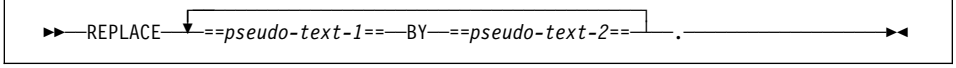
The REPLACE statement is used to replace source program text.

A REPLACE statement can occur anywhere in the source program where a character-string can occur. It must be preceded by a separator period except when it is the first statement in a separately compiled program. It must be terminated by a separator period.

The REPLACE statement provides a means of applying a change to an entire COBOL source program, or part of a source program, without manually having to find and modify all places that need to be changed. It is an easy method of doing simple string substitutions. It is similar in action to the REPLACING phrase of the COPY statement, except that it acts on the entire source program, not just on the text in COPY libraries.

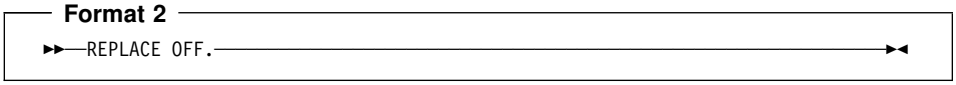
If the word REPLACE appears in a comment-entry or in the place where a comment-entry can appear, it is considered part of the comment-entry.

**Format 1**




Each matched occurrence of pseudo-text-1 in the source program is replaced by the corresponding pseudo-text-2.

**Format 2**




Any text replacement currently in effect is discontinued with the format 2 form of REPLACE. If format 2 is not specified, a given occurrence of the REPLACE statement is in effect from the point at which it is specified until the next occurrence of the statement or the end of the separately compiled program, respectively.

**pseudo-text-1**
Must contain one or more text words. Character-strings can be continued.

x As an IBM extension, pseudo-text-1 can consist entirely of a separator comma or a
x separator semicolon.

**pseudo-text-2**
Can contain zero, one, or more text words. Character strings can be continued.

Any individual character-string within pseudo-text can be up to 322 characters long.

**Note:** Characters outside the standard COBOL character set should not appear in pseudo-text, other than as part of nonnumeric literals, comment lines, or comment-entries.

x The REPLACE statement can be used with DBCS literals and DBCS or EUC names.

x Pseudo-text can contain DBCS or EUC character-strings, but the characters cannot be
x continued across lines.

The compiler processes REPLACE statements in a source program after the processing of any COPY statements. COPY must be processed first, to assemble a complete source program. Then REPLACE can be used to modify that program, performing simple string substitution. REPLACE statements can themselves contain COPY statements.

The text produced as a result of the processing of a REPLACE statement must not contain a REPLACE statement.

## Continuation Rules for Pseudo-text

The character-strings and separators comprising pseudo-text can start in either area A or area B. If, however, there is a hyphen in the indicator area of a line which follows the opening pseudo-text delimiter, area A of the line must be blank; and the normal rules for continuation of lines apply to the formation of text words. (See “Continuation Lines” on page 37.)

## Comparison Operation

The comparison operation to determine text replacement starts with the leftmost source program text word following the REPLACE statement, and with the first pseudo-text-1. Pseudo-text-1 is compared to an equivalent number of contiguous source program text words. Pseudo-text-1 matches the source program 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 program text words.

For purposes of matching, each occurrence of a separator comma, semicolon, and space, and each sequence of one or more space separators is considered to be a single space.

x However, when pseudo-text-1 consists solely of a separator comma or semicolon, it
x participates in the match as a text word (in this case, the space following the comma or
x semicolon separator can be omitted).

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

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

## REPLACE Statement

Whenever a match occurs between pseudo-text-1 and the source program text, the corresponding pseudo-text-2 replaces the matched text in the source program. The source program text word immediately following the rightmost text word that participated in the match is then considered as the leftmost source program text word. The comparison cycle starts again with the first occurrence of pseudo-text-1.

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

## REPLACE Statement Notes

Comment lines or blank lines occurring in the source program text and in pseudo-text-1 are ignored for purposes of matching. The sequence of text words in the source program text and in pseudo-text-1 is determined by the rules for reference format (see "Reference Format" on page 34). Comment lines or blank lines in pseudo-text-2 are placed into the resultant program unchanged whenever pseudo-text-2 is placed into the source program as a result of text replacement. Comment lines or blank lines appearing in source program text are retained unchanged with the following exception: a comment line or blank line in source program text is not retained if that comment line or blank line appears within the sequence of text words that match pseudo-text-1.

x Lines containing *CONTROL (*CBL), EJECT, SKIP1/2/3, or TITLE statements can
x occur in source program text. Such lines are treated as comment lines during
x REPLACE statement processing.

Debugging lines are permitted in pseudo-text. Text words within a debugging line participate in the matching rules as if the D did not appear in the indicator area.

When a REPLACE statement is specified on a debugging line, the statement is treated as if the D did not appear in the indicator area.

After all COPY and REPLACE statements have been processed, a debugging line will be considered to have all the characteristics of a comment line, if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

Except for COPY and REPLACE statements, the syntactic correctness of the source program text cannot be determined until after all COPY and REPLACE statements have been completely processed.

Text words inserted into the source program as a result of processing a REPLACE statement are placed in the source program according to the rules for reference format. When inserting text words of pseudo-text-2 into the source program, additional spaces are introduced only between text words where there already exists a space (including the assumed space between source lines).

If additional lines are introduced into the source program as a result of the processing of REPLACE statements, the indicator area of the introduced lines contains the same

character as the line on which the text being replaced begins, unless that line contains a hyphen, in which case the introduced line contains a space.

If any literal within pseudo-text-2 is of a length too great to be accommodated on a single line without continuation to another line in the resultant program and the literal is not being placed on a debugging line, additional continuation lines are introduced that contain the remainder of the literal. If replacement requires the continued literal to be continued on a debugging line, the program is in error.

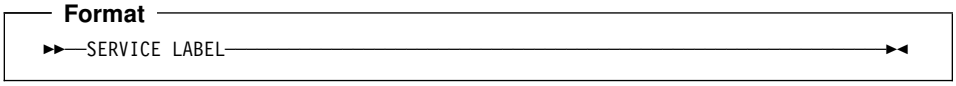
**Note:** Each word in pseudo-text-2 that is to be placed into the resultant program begins in the same area of the resultant program as it appears in pseudo-text-2.

## SERVICE LABEL Statement

WRKSTN Under AIX, OS/2, and Windows, the SERVICE LABEL statement is treated as a comment.

This statement is generated by the CICS preprocessor to indicate control flow. It is not intended for general use.

**Format**




The SERVICE LABEL statement can appear only in the Procedure Division, not in the Declaratives Section.

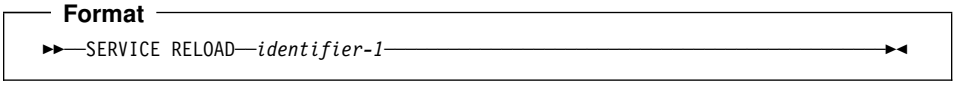
At the statement following the SERVICE LABEL statement, all registers that might no longer be valid are reloaded.

See *IBM COBOL for MVS & VM Programming Guide* for more information.

## SERVICE RELOAD Statement

The SERVICE RELOAD statement is treated as a comment.

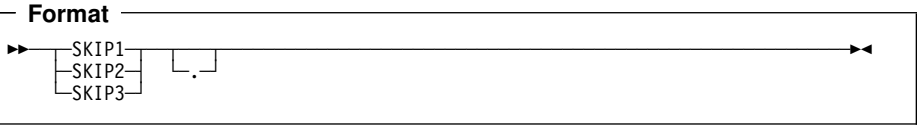
**Format**



## SKIP1/2/3 Statements

The SKIP1/2/3 statements specify blank lines that the compiler should add when printing the source listing. SKIP statements have no effect on the compilation of the source program itself.

**Format**




**SKIP1**
Specifies a single blank line to be inserted in the source listing.

**SKIP2**
Specifies two blank lines to be inserted in the source listing.

**SKIP3**
Specifies three blank lines to be inserted in the source listing.

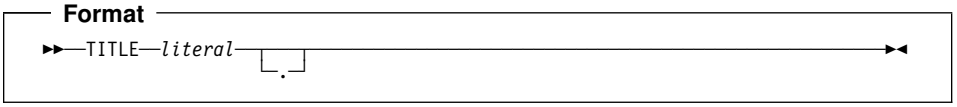
SKIP1, SKIP2, or SKIP3 can be written anywhere in either Area A or Area B, and can be terminated with a separator period. It must be the only statement on the line.

The SKIP1/2/3 statement must be embedded in a program source. For example, in the case of batch jobs, the SKIP1/2/3 statement must be placed between the CBL (PROCESS) statement and the end of the program (or the END PROGRAM header, if specified).

## TITLE Statement

The TITLE statement specifies a title to be printed at the top of each page of the source listing produced during compilation. If no TITLE statement is found, a title containing the identification of the compiler and the current release level is generated. The title is left-justified on the title line.

**Format**




**literal**

Must be nonnumeric and can be followed by a separator period.

Can be a DBCS or EUC literal.

Must not be a figurative constant.

In addition to the default or chosen title, the right side of the title line contains:

- Name of the program from the PROGRAM-ID paragraph for the outermost program (This space is blank on pages preceding the PROGRAM-ID paragraph for the outermost program.)
- Current page number
- Date and time of compilation

The TITLE statement:

- Forces a new page immediately, if the SOURCE compiler option is in effect
- Is not printed on the source listing
- Has no other effect on compilation
- Has no effect on program execution
- Cannot be continued on another line
- Can appear anywhere in any of the divisions

A title line is produced for each page in the listing produced by the LIST option. This title line uses the last TITLE statement found in the source statements or the default.

The word TITLE can begin in either Area A or Area B.

The TITLE statement must be embedded in a program source. For example, in the case of batch jobs, the TITLE statement must be placed between the CBL (PROCESS) statement and the end of the program (or the END PROGRAM header, if specified).

No other statement can appear on the same line as the TITLE statement.

## USE Statement

The formats for the USE statement are:

EXCEPTION/ERROR declarative
x LABEL declarative (MVS and VM Only)
DEBUGGING declarative

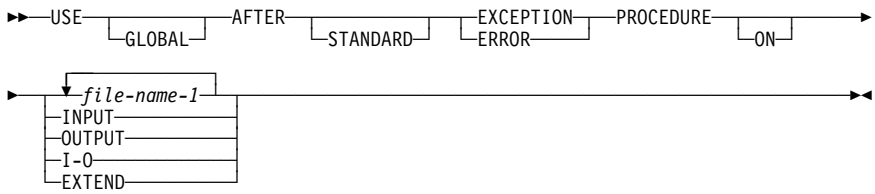
For general information on declaratives, see “Declaratives” on page 220.

## EXCEPTION/ERROR Declarative

The EXCEPTION/ERROR declarative specifies procedures for input/output exception or error handling that are to be executed in addition to the standard system procedures.

The words EXCEPTION and ERROR are synonymous and can be used interchangeably.

**Format 1—USE (EXCEPTION ERROR Declarative)**




**file-name-1**
Valid for all files. When this option is specified, the procedure is executed only for the file(s) named. No file-name can refer to a sort or merge file. For any given file, only one EXCEPTION/ERROR procedure can be specified; thus, file-name specification must not cause simultaneous requests for execution of more than one EXCEPTION/ERROR procedure.

A USE AFTER EXCEPTION/ERROR declarative statement specifying the name of a file takes precedence over a declarative statement specifying the open mode of the file.

**INPUT**
Valid for all files. When this option is specified, the procedure is executed for all files opened in INPUT mode or in the process of being opened in INPUT mode that get an error.

**OUTPUT**
Valid for all files. When this option is specified, the procedure is executed for all files opened in OUTPUT mode or in the process of being opened in OUTPUT mode that get an error.

**I-O** Valid for all direct-access files. When this option is specified, the procedure is executed for all files opened in I-O mode or in the process of being opened in I-O mode that get an error.

**EXTEND**
Valid for all files. When this option is specified, the procedure is executed for all files opened in EXTEND mode or in the process of being opened in EXTEND mode that get an error.

The EXCEPTION/ERROR procedure is executed:

- Either after completing the system-defined input/output error routine, or
- Upon recognition of an INVALID KEY or AT END condition when an INVALID KEY or AT END phrase has not been specified in the input/output statement, or
- Upon recognition of an IBM-defined condition that causes status key 1 to be set to 9. (See "Status Key" on page 254.)

After execution of the EXCEPTION/ERROR procedure, control is returned to the invoking routine in the input/output control system. If the input/output status value does not indicate a critical 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.

The EXCEPTION/ERROR procedures are activated when an input/output error occurs during execution of a READ, WRITE, REWRITE, START, OPEN, CLOSE, or DELETE statement. To determine what conditions are errors see "Common Processing Facilities" on page 254.

Within a declarative procedure, there must be no reference to any non-declarative procedures. In the non-declarative portion of the program, there must be no reference to procedure-names that appear in an EXCEPTION/ERROR declarative procedure, except that PERFORM statements can refer to an EXCEPTION/ERROR procedure or to procedures associated with it.

As IBM extensions to the COBOL 85 Standard, the following apply to declarative procedures:

For AIX, OS/2, Windows, MVS, and VM:

- A declarative procedure can be performed from a nondeclarative procedure.

Additionally for MVS and VM:

- A nondeclarative procedure can be performed from a declarative procedure.
- A declarative procedure can be referenced in a GO TO statement in a declarative procedure.
- A nondeclarative procedure can be referenced in a GO TO statement in a declarative procedure.

Within an EXCEPTION/ERROR declarative procedure, no statement should be included that would cause execution of a USE procedure that had been previously invoked and had not yet returned control to the invoking routine.

## USE Statement

You can include a statement that executes a previously invoked USE procedure that is still in control. However, to avoid an infinite loop, you must be sure that there is an eventual exit at the bottom.

EXCEPTION/ERROR procedures can be used to check the status key values whenever an input/output error occurs.

## Precedence Rules for Nested Programs

Special precedence rules are followed when programs are contained within other programs. In applying these rules, only the first qualifying declarative that is selected for execution must satisfy the rules for execution of that declarative. The order of precedence for selecting a declarative is:

1. A file-specific declarative (that is, a declarative of the form USE AFTER ERROR ON file-name-1) within the program that contains the statement that caused the qualifying condition.
2. A mode-specific declarative (that is, a declarative of the form USE AFTER ERROR ON INPUT) within the program that contains the statement that caused the qualifying condition.
3. A file-specific declarative that specifies the GLOBAL phrase and is within the program directly containing the program that was last examined for a qualifying declarative.
4. A mode-specific declarative that specifies the GLOBAL phrase and is within the program directly containing the program that was last examined for a qualifying condition.

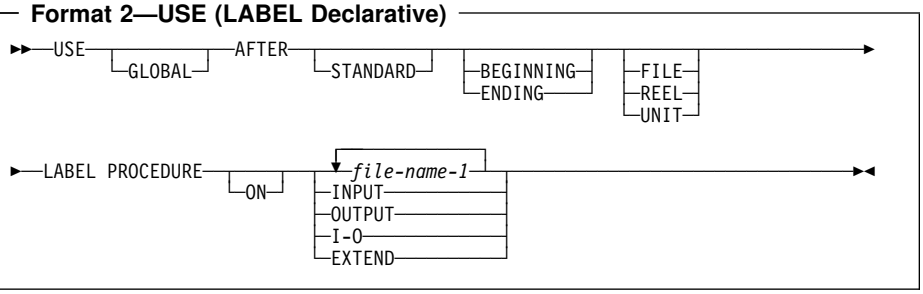
Steps 3. and 4. are repeated until the last examined program is the outermost program, or until a qualifying declarative has been found.

## LABEL Declarative

The LABEL declarative provides user label-handling procedures.

WRKSTN Under AIX, OS/2, and Windows, USE...AFTER...LABEL PROCEDURE is not supported. If encountered, they are ignored and a warning message is issued.

**Format 2—USE (LABEL Declarative)**



**AFTER**
User labels follow standard file labels, and are to be processed.

The labels must be listed as data-names in the LABEL RECORDS clause in the file description entry for the file, and must be described as level-01 data items subordinate to the file entry.

If neither BEGINNING nor ENDING is specified, the designated procedures are executed for both beginning and ending labels.

If FILE, REEL, or UNIT is not included, the designated procedures are executed both for REEL or UNIT, whichever is appropriate, and for FILE labels.

**FILE**
The designated procedures are executed at beginning-of-file (on the first volume) and/or at end-of-file (on the last volume) only.

**REEL**
The designated procedures are executed at beginning-of-volume (on each volume except the first) and/or at end-of-volume (on each volume except the last).

The REEL option is not applicable to direct-access files.

**UNIT**
The designated procedures are executed at beginning-of-volume (on each volume except the first) and/or at end-of-volume (on each volume except the last).

The UNIT phrase is not applicable to files in the random access mode, because only FILE labels are processed in this mode.

**file-name-1**
Can appear in different specific arrangements of the format. However, appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE declarative.

**file-name-1** must not represent a sort file.

If the **file-name-1** option is used, the file description entry for file-name must not specify a LABEL RECORDS ARE OMITTED clause.

When the INPUT, OUTPUT, or I-O options are specified, user label procedures are executed as follows:

- When INPUT is specified, only for files opened as input
- When OUTPUT is specified, only for files opened as output
- When I-O is specified, only for files opened as I-O
- When EXTEND is specified, only for files opened EXTEND

If the INPUT, OUTPUT, or I-O phrase is specified, and an input, output, or I-O file, respectively, is described with a LABEL RECORDS ARE OMITTED clause, the USE procedures do not apply. The standard system procedures are performed:

- Before the beginning or ending input label check procedure is executed
- Before the beginning or ending output label is created

- After the beginning or ending output label is created, but before it is written on tape
- Before the beginning or ending input-output label check procedure is executed

Within the procedures of a USE declarative in which the USE sentence specifies an option other than **file-name**, references to common label items need not be qualified by a file-name. A common label item is an elementary data item that appears in every label record of the program, but does not appear in any data records of this program. Such items must have identical descriptions and positions within each label record.

Within a Declarative Section there must be no reference to any non-declarative procedure. Conversely, in the non-declarative portion there must be no reference to procedure-names that appear in the Declarative Section, except that the PERFORM statement can refer to a USE procedure, or to procedures associated with it.

The exit from a Declarative Section is inserted by the compiler following the last statement in the section. All logical program paths within the section must lead to the exit point.

There is one exception: A special exit can be specified by the statement GO TO MORE-LABELS. When an exit is made from a Declarative Section by means of this statement, the system will do one of the following:

1. Write the current beginning or ending label and then reenter the USE section at its beginning for further creating of labels. After creating the last label, the user must exit by executing the last statement of the section.
2. Read an additional beginning or ending label, and then reenter the USE section at its beginning for further checking of labels. When processing user labels, the section will be reentered only if there is another user label to check. Hence, there need not be a program path that flows through the last statement in the section.

If a GO TO MORE-LABELS statement is not executed for a user label, the declarative section is not reentered to check or create any immediately succeeding user labels.

## DEBUGGING Declarative

Debugging sections are permitted only in the outermost program; they are not valid in nested programs. Debugging sections are never triggered by procedures contained in nested programs.

Debugging sections are not permitted in:

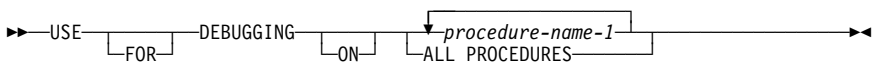
- A program or method defined with the RECURSIVE attribute
- A program compiled with the THREAD compiler option (Workstation only)

The WITH DEBUGGING MODE clause of the SOURCE compiler statement activates all debugging sections and lines that have been compiled into the object program. See Appendix C, “Source Language Debugging” on page 529, for additional details.

When the debugging mode is suppressed by not specifying that option of the SOURCE compiler, any USE FOR DEBUGGING declarative procedures and all debugging lines are inhibited.

Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

**Format 3—USE (DEBUGGING Declarative)**




**USE FOR DEBUGGING**

All debugging statements must be written together in a section immediately after the DECLARATIVES header.

Except for the USE FOR DEBUGGING sentence itself, within the debugging procedure there must be no reference to any non-declarative procedures.

**procedure-name-1**

Must not be defined in a debugging session.

Table 48 shows, for each valid option, the points during program execution when the USE FOR DEBUGGING procedures are executed.

Any given procedure-name can appear in only one USE FOR DEBUGGING sentence, and only once in that sentence. All procedures must appear in the outermost program.

**ALL PROCEDURES**

Procedure-name-1 must not be specified in any USE FOR DEBUGGING sentences. The ALL PROCEDURES phrase can be specified only once in a program. Only the procedures contained in the outermost program will trigger execution of the debugging section.

*Table 48. Execution of Debugging Declaratives*

| USE FOR DEBUGGING Operand | Upon execution of the following, the USE FOR DEBUGGING procedures are executed immediately |
|---|---|
| procedure-name-1 | Before each execution of the named procedure |
| | After the execution of an ALTER statement referring to the named procedure |
| ALL PROCEDURES | Before each execution of every nondebugging procedure in the outermost program |
| | After the execution of every ALTER statement in the outermost program (except ALTER statements in declarative procedures) |

# Compiler Directives

A **Compiler Directive** is a statement that causes the compiler to take a specific action during compilation.

Currently, CALLINTERFACE is the only compiler directive supported.

## CALLINTERFACE

The CALLINTERFACE directive specifies the interface convention for CALL...USING statements. The convention specified stays in effect until another CALLINTERFACE specification is made in the source program.

MVS/VM Under MVS and VM, the CALLINTERFACE directive is not supported.

**Format**

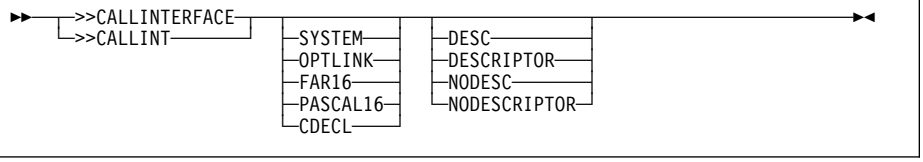



*Table 49. CALLINTERFACE Options supported by Platform*

| AIX | OS/2 | Windows |
|---|---|---|
| SYSTEM | SYSTEM | SYSTEM |
| | OPTLINK | OPTLINK |
| | FAR16 | CDECL |
| | PASCAL16 | |
| DESC | DESC | DESC |
| NODESC | NODESC | NODESC |

**Note:** If you specify a suboption that is not applicable to the platform, the entire CALLINTERFACE directive is ignored for that platform. For example, if you specify `>>CALLINT CDECL` for an OS/2 program, it is ignored.

For more information on which suboptions are in effect for multiple directive and compiler option specifications, see “Precedence of Sub Options” on page 516.

**SYSTEM**
Specifies that the system linkage convention of the platform is used as the call interface convention.

**OPTLINK**
Specifies that the _Optlink calling convention as defined by C Set++ is used as the call interface convention.


© Copyright IBM Corp. 1991, 1996

x **FAR16**
x Specifies that the _FAR16_CDecl calling convention as defined by C Set++ is used
x as the call interface convention.

| **PASCAL16**
| Specifies that the _FAR16_Pascal calling convention as defined by C Set++ is
| used as the call interface convention.

| **CDECL**
| Specifies that the CDECL calling convention as defined by MicroSoft Visual C++ is
| used as the call interface convention.

| **DESC, DESCRIPTOR**
| Indicates that an argument descriptor is passed for each argument on a CALL
| statement.

| **NODESC, NODESCRIPTOR**
| Indicates that no argument descriptors are passed for any arguments on a CALL
| statement. NODESC/NODESCRIPTOR is the default.

x Specify CALLINTERFACE only in the Procedure Division.

x The positions of CALL statements relative to the >>CALLINTERFACE directive are
x recognized following any processing of COPY and REPLACE statements. For
x example, CALL statements and >>CALLINTERFACE statements in COPY text are
x processed by the rules specified for the directive.

## x Syntax and General Rules

x • You must specify a >>CALLINTERFACE on a line by itself, in Area B.

x • You cannot specify >>CALLINTERFACE:

x – Within a source text manipulation sentence (for example, COPY or
x REPLACING)
x – Between the lines of a continued character string
x – On a debugging line
x – In the middle of a COBOL statement

x • The >>CALLINTERFACE specification is limited to the current program.

x • The REPLACE statement and REPLACE phrase of the COPY statement do not
x affect the CALLINTERFACE specification.

## x Difference Between the Directive and Compiler Option

x You can indicate which calling convention you want by using either the
x CALLINTERFACE directive or the CALLINT compiler option. Use the directive when
x you want to use more than one call convention for the CALL statements in a compila-
x tion unit. Use the compiler option when you want to use the same call convention for
x the entire compilation unit.

x

### Precedence of Sub Options

If you specify both the CALLINTERFACE directive (with suboptions) and the CALLINT compiler option, the directive overrides the compiler option specification for the statements following the directive within a source program.

If you specify the CALLINTERFACE directive without any suboptions, the CALLINT compiler option specification is in effect.

If you specify only the DESC/NODESC suboption, the calling convention in effect is the convention specified in the CALLINT compiler option. (DESC/NODESC are options only for the CALLINTERFACE directive. They are not available in the CALLINT compiler option.) For example, if the CALLINT compiler option is set to CALLINT SYSTEM, given the following directives:

```
(Section A)
⋮
>>CALLINTERFACE OPT
(Section B)
⋮
>>CALLINTERFACE DESC
(Section C)
```

the following specifications are in effect:

- Section A—SYSTEM
- Section B—OPT
- Section C—SYSTEM DESC

# Appendixes

© Copyright IBM Corp. 1991, 1996

# Appendix A. Compiler Limits

The following table lists the compiler limits for IBM COBOL for MVS & VM programs running under MVS/ESA. Compiler limits for programs running under AIX, OS/2, and Windows are also included. Other operating systems may impose further limits. The numbers are **guidelines** to the limits.

*Table 50 (Page 1 of 4). Compiler Limits*

| Language Element | Compiler Limit |
|---|---|
| Size of program | 999,999 lines |
| Size of file record size (AIX, OS/2, and Windows) | 64K |
| Number of literals | 4,194,303[1] |
| Total length of literals | 4,194,303 bytes[1] |
| Reserved Word Table entries | 1536 |
| COPY REPLACING ... BY ... (items per COPY statement) | N/A |
| Number of COPY libraries | N/A |
| Block size of COPY library | 32,767 bytes |
| Identification Division | |
| Environment Division | |
| Configuration Section | |
| SPECIAL-NAMES paragraph | |
| function-name IS | 18 |
| UPSI-n ... (switches) | 0-7 |
| alphabet-name IS ... | N/A |
| literal THRU/ALSO ... | 256 |
| Input-Output Section | |
| FILE-CONTROL paragraph | |
| SELECT file-name ... | 65,535 |
| ASSIGN system-name ... | N/A[2] |
| ALTERNATE RECORD KEY data-name ... | 253 |
| RECORD KEY length | N/A[3] |
| RESERVE integer (buffers) | 255[4] |
| I-O-CONTROL paragraph | |
| RERUN ON system-name ... | 32,767 |
| integer RECORDS | 16,777,215 |

*Table 50 (Page 2 of 4). Compiler Limits*

| Language Element | Compiler Limit |
|---|---|
| SAME RECORD AREA | 255 |
| FOR file-name ... | 255 |
| SAME SORT/MERGE AREA | N/A[2] |
| MULTIPLE FILE ... file-name | N/A[2] |
| Data Division | |
| File Section | |
| FD file-name ... | 65,535 |
| LABEL data-name ... (if no optional clauses) | 255 |
| Label record length | 80 bytes |
| DATA RECORD dnm ... | N/A[2] |
| BLOCK CONTAINS integer | 1,048,575[5] |
| RECORD CONTAINS integer | 1,048,575[5] |
| Item length | 1,048,575 bytes[5] |
| LINAGE clause values | 2**32 (4-byte binary numbers) |
| SD file-name ... | 65,535 |
| DATA RECORD dnm ... | N/A[2] |
| Sort record length | 32,751 bytes |
| Working-Storage Section | |
| items without the EXTERNAL attribute | 134,217,727 bytes |
| items with the EXTERNAL attribute | 134,217,727 bytes |
| 77 data-names | 16,777,215 bytes |
| 01-49 data-names | 16,777,215 bytes |
| 88 condition-name ... | N/A |
| VALUE literal ... | N/A |
| 66 RENAMES ... | N/A |
| PICTURE character-string | 30 |
| Numeric item digit positions | 18 |
| Num-edit character positions | 249 |
| PICTURE replication ( ) | 16,777,215 |
| PIC repl (editing) | 32,767 |
| DBCS Picture replication ( ) | 8,388,607 |
| Group item size: File Section | 1,048,575 bytes |
| Elementary item size | 16,777,215 bytes |

## Compiler Limits

*Table 50 (Page 3 of 4). Compiler Limits*

| Language Element | Compiler Limit |
|---|---|
| VALUE initialization (Total length of all value literals) | 16,777,215 bytes |
| OCCURS integer | 16,777,215 |
| Total number of ODOs | 4,194,303[1] |
| Table size | 16,777,215 bytes |
| Table element size | 8,388,607 bytes |
| ASC/DES KEY ... (per OCCURS clause) | 12 KEYS |
| Total length | 256 bytes |
| INDEXED BY ... (index names) (per OCCURS clause) | 12 |
| Total num of indexes (index names) | 65,535 |
| Size of relative index | 32,765 |
| Linkage Section | 134,217,727 bytes |
| Total 01 + 77 (data items) | N/A |
| Procedure Division | |
| Procedure + constant area | 4,194,303 bytes[1] |
| USING identifier ... | 32,767 |
| Procedure-names | 1,048,575[1] |
| Subscripted data-names per verb | 32,767 |
| Verbs per line (TEST) | 7 |
| ADD identifier ... | N/A |
| ALTER pn1 TO pn2 ... | 4,194,303[1] |
| CALL ... BY CONTENT id | 2,147,483,647 bytes |
| CALL id/lit USING id/lit... | 16380 (MVS and VM) 500 (AIX, OS/2, and Windows) |
| CALL literal ... | 4,194,303[1] |
| Active programs in run unit | 32,767 |
| number of names called (DYN) | N/A |
| CANCEL id/lit ... | N/A |
| CLOSE file-name ... | N/A |
| COMPUTE identifier ... | N/A |
| DISPLAY id/lit ... | N/A |
| DIVIDE identifier ... | N/A |
| ENTRY USING id/lit ... | N/A |
| EVALUATE ... subjects | 64 |
| EVALUATE ... WHEN clauses | 256 |

*Table 50 (Page 4 of 4). Compiler Limits*

| Language Element | Compiler Limit |
|---|---|
| GO pn ... DEPENDING | 255 |
| INSPECT TALLY/REPL clauses | N/A |
| MERGE file-name ASC/DES KEY ... | N/A |
| Total key length | 4,092 bytes[6] |
| USING file-name ... | 16[7] |
| MOVE id/lit TO id ... | N/A |
| MULTIPLY identifier ... | N/A |
| OPEN file-name | N/A |
| PERFORM | 4,194,303 |
| SEARCH ... WHEN ... | N/A |
| SET index/id ... TO | N/A |
| SET index ... UP/DOWN | N/A |
| SORT file-name ASC/DES KEY | N/A |
| Total key length | 4,092 bytes[6] |
| USING file-name ... | 16[7] |
| STRING identifier ... | N/A |
| DELIMITED id/lit ... | N/A |
| UNSTRING DELIMITED id/lit OR id/lit ... | 255 |
| UNSTRING INTO id/lit ... | N/A |
| USE ... ON file-name ... | N/A |

**Table Notes (MVS and VM Only):**

1 Items included in 4,194,303 byte limit for procedure plus constant area.
2 Treated as comment; there is no limit.
3 No compiler limit, but VSAM limits it to 255 bytes.
4 QSAM
5 Compiler limit shown, but QSAM limits it to 32,767 bytes.
6 For QSAM and VSAM, the limit is 4088 bytes if EQUALS is coded on the OPTION control statement.
7 SORT limit for QSAM and VSAM.

# Appendix B. EBCDIC and ASCII Collating Sequences

The ascending collating sequences for both the EBCDIC (Extended Binary Coded Decimal Interchange Code) and ASCII (American National Standard Code for Information Interchange) character sets are shown in this appendix. In addition to the symbol and meaning for each character, the ordinal number (beginning with 1), decimal representation, and hexadecimal representation are given.

## EBCDIC Collating Sequence

*Table 51 (Page 1 of 4). EBCDIC Collating Sequence*

| Ordinal Number | Symbol | Meaning | Decimal Representation | Hex Representation |
|---|---|---|---|---|
| 65 | b | Space | 64 | 40 |
| ⋮ | | | | |
| 75 | ¢ | Cent sign | 74 | 4A |
| 76 | . | Period, decimal point | 75 | 4B |
| 77 | < | Less than sign | 76 | 4C |
| 78 | ( | Left parenthesis | 77 | 4D |
| 79 | + | Plus sign | 78 | 4E |
| 80 | \| | Vertical bar, Logical OR | 79 | 4F |
| 81 | & | Ampersand | 80 | 50 |
| ⋮ | | | | |
| 91 | ! | Exclamation point | 90 | 5A |
| 92 | $ | Dollar sign | 91 | 5B |
| 93 | * | Asterisk | 92 | 5C |
| 94 | ) | Right parenthesis | 93 | 5D |
| 95 | ; | Semicolon | 94 | 5E |
| 96 | ¬ | Logical NOT | 95 | 5F |
| 97 | - | Minus, hyphen | 96 | 60 |
| 98 | / | Slash | 97 | 61 |
| ⋮ | | | | |
| 108 | , | Comma | 107 | 6B |
| 109 | % | Percent sign | 108 | 6C |
| 110 | _ | Underscore | 109 | 6D |
| 111 | > | Greater than sign | 110 | 6E |
| 112 | ? | Question mark | 111 | 6F |
| ⋮ | | | | |


© Copyright IBM Corp. 1991, 1996

## EBCDIC Collating Sequence

*Table 51 (Page 2 of 4). EBCDIC Collating Sequence*

| Ordinal Number | Symbol | Meaning | Decimal Representation | Hex Representation |
|---|---|---|---|---|
| 123 | : | Colon | 122 | 7A |
| 124 | # | Number sign, pound sign | 123 | 7B |
| 125 | @ | At sign | 124 | 7C |
| 126 | ' | Apostrophe, prime sign | 125 | 7D |
| 127 | = | Equal sign | 126 | 7E |
| 128 | " | Quotation marks | 127 | 7F |
| ⋮ | | | | |
| 130 | a | | 129 | 81 |
| 131 | b | | 130 | 82 |
| 132 | c | | 131 | 83 |
| 133 | d | | 132 | 84 |
| 134 | e | | 133 | 85 |
| 135 | f | | 134 | 86 |
| 136 | g | | 135 | 87 |
| 137 | h | | 136 | 88 |
| 138 | i | | 137 | 89 |
| ⋮ | | | | |
| 146 | j | | 145 | 91 |
| 147 | k | | 146 | 92 |
| 148 | l | | 147 | 93 |
| 149 | m | | 148 | 94 |
| 150 | n | | 149 | 95 |
| 151 | o | | 150 | 96 |
| 152 | p | | 151 | 97 |
| 153 | q | | 152 | 98 |
| 154 | r | | 153 | 99 |
| ⋮ | | | | |
| 163 | s | | 162 | A2 |
| 164 | t | | 163 | A3 |
| 165 | u | | 164 | A4 |
| 166 | v | | 165 | A5 |
| 167 | w | | 166 | A6 |
| 168 | x | | 167 | A7 |
| 169 | y | | 168 | A8 |

## EBCDIC Collating Sequence

*Table 51 (Page 3 of 4). EBCDIC Collating Sequence*

| Ordinal Number | Symbol | Meaning | Decimal Representation | Hex Representation |
|---|---|---|---|---|
| 170 | z | | 169 | A9 |
| ⋮ | | | | |
| 194 | A | | 193 | C1 |
| 195 | B | | 194 | C2 |
| 196 | C | | 195 | C3 |
| 197 | D | | 196 | C4 |
| 198 | E | | 197 | C5 |
| 199 | F | | 198 | C6 |
| 200 | G | | 199 | C7 |
| 201 | H | | 200 | C8 |
| 202 | I | | 201 | C9 |
| ⋮ | | | | |
| 210 | J | | 209 | D1 |
| 211 | K | | 210 | D2 |
| 212 | L | | 211 | D3 |
| 213 | M | | 212 | D4 |
| 214 | N | | 213 | D5 |
| 215 | O | | 214 | D6 |
| 216 | P | | 215 | D7 |
| 217 | Q | | 216 | D8 |
| 218 | R | | 217 | D9 |
| ⋮ | | | | |
| 227 | S | | 226 | E2 |
| 228 | T | | 227 | E3 |
| 229 | U | | 228 | E4 |
| 230 | V | | 229 | E5 |
| 231 | W | | 230 | E6 |
| 232 | X | | 231 | E7 |
| 233 | Y | | 232 | E8 |
| 234 | Z | | 233 | E9 |
| ⋮ | | | | |
| 241 | 0 | | 240 | F0 |
| 242 | 1 | | 241 | F1 |
| 243 | 2 | | 242 | F2 |

*Table 51 (Page 4 of 4). EBCDIC Collating Sequence*

| Ordinal Number | Symbol | Meaning | Decimal Representation | Hex Representation |
|---|---|---|---|---|
| 244 | 3 | | 243 | F3 |
| 245 | 4 | | 244 | F4 |
| 246 | 5 | | 245 | F5 |
| 247 | 6 | | 246 | F6 |
| 248 | 7 | | 247 | F7 |
| 249 | 8 | | 248 | F8 |
| 250 | 9 | | 249 | F9 |

## ASCII Code Values

## US English ASCII Code Page (ISO 646)

*Table 52 (Page 1 of 3). ASCII Collating Sequence*

| Ordinal Number | Symbol | Meaning | Decimal Representation | Hex Representation |
|---|---|---|---|---|
| 1 | | Null | 0 | 0 |
| ⋮ | | | | |
| 33 | b | Space | 32 | 20 |
| 34 | ! | Exclamation point | 33 | 21 |
| 35 | " | Quotation mark | 34 | 22 |
| 36 | # | Number sign | 35 | 23 |
| 37 | $ | Dollar sign | 36 | 24 |
| 38 | % | Percent sign | 37 | 25 |
| 39 | & | Ampersand | 38 | 26 |
| 40 | ' | Apostrophe, prime sign | 39 | 27 |
| 41 | ( | Opening parenthesis | 40 | 28 |
| 42 | ) | Closing parenthesis | 41 | 29 |
| 43 | * | Asterisk | 42 | 2A |
| 44 | + | Plus sign | 43 | 2B |
| 45 | , | Comma | 44 | 2C |
| 46 | - | Hyphen, minus | 45 | 2D |
| 47 | . | Period, decimal point | 46 | 2E |
| 48 | / | Slant | 47 | 2F |
| 49 | 0 | | 48 | 30 |
| 50 | 1 | | 49 | 31 |
| 51 | 2 | | 50 | 32 |
| 52 | 3 | | 51 | 33 |
| 53 | 4 | | 52 | 34 |
| 54 | 5 | | 53 | 35 |
| 55 | 6 | | 54 | 36 |
| 56 | 7 | | 55 | 37 |
| 57 | 8 | | 56 | 38 |
| 58 | 9 | | 57 | 39 |
| 59 | : | Colon | 58 | 3A |
| 60 | ; | Semicolon | 59 | 3B |
| 61 | < | Less than sign | 60 | 3C |
| 62 | = | Equal sign | 61 | 3D |

*Table 52 (Page 2 of 3). ASCII Collating Sequence*

| Ordinal Number | Symbol | Meaning | Decimal Representation | Hex Representation |
|---|---|---|---|---|
| 63 | > | Greater than sign | 62 | 3E |
| 64 | ? | Question mark | 63 | 3F |
| 65 | @ | Commercial At sign | 64 | 40 |
| 66 | A | | 65 | 41 |
| 67 | B | | 66 | 42 |
| 68 | C | | 67 | 43 |
| 69 | D | | 68 | 44 |
| 70 | E | | 69 | 45 |
| 71 | F | | 70 | 46 |
| 72 | G | | 71 | 47 |
| 73 | H | | 72 | 48 |
| 74 | I | | 73 | 49 |
| 75 | J | | 74 | 4A |
| 76 | K | | 75 | 4B |
| 77 | L | | 76 | 4C |
| 78 | M | | 77 | 4D |
| 79 | N | | 78 | 4E |
| 80 | O | | 79 | 4F |
| 81 | P | | 80 | 50 |
| 82 | Q | | 81 | 51 |
| 83 | R | | 82 | 52 |
| 84 | S | | 83 | 53 |
| 85 | T | | 84 | 54 |
| 86 | U | | 85 | 55 |
| 87 | V | | 86 | 56 |
| 88 | W | | 87 | 57 |
| 89 | X | | 88 | 58 |
| 90 | Y | | 89 | 59 |
| 91 | Z | | 90 | 5A |
| 92 | [ | Opening bracket | 91 | 5B |
| 93 | \ | Reverse slant | 92 | 5C |
| 94 | ] | Closing bracket | 93 | 5D |
| 95 | ^ | Caret | 94 | 5E |
| 96 | _ | Underscore | 95 | 5F |

## ASCII Code Values

*Table 52 (Page 3 of 3). ASCII Collating Sequence*

| Ordinal Number | Symbol | Meaning | Decimal Representation | Hex Representation |
|---|---|---|---|---|
| 97 | ` | Grave Accent | 96 | 60 |
| 98 | a | | 97 | 61 |
| 99 | b | | 98 | 62 |
| 00 | c | | 99 | 63 |
| 01 | d | | 100 | 64 |
| 02 | e | | 101 | 65 |
| 03 | f | | 102 | 66 |
| 04 | g | | 103 | 67 |
| 05 | h | | 104 | 68 |
| 06 | i | | 105 | 69 |
| 07 | j | | 106 | 6A |
| 08 | k | | 107 | 6B |
| 09 | l | | 108 | 6C |
| 10 | m | | 109 | 6D |
| 11 | n | | 110 | 6E |
| 12 | o | | 111 | 6F |
| 13 | p | | 112 | 70 |
| 14 | q | | 113 | 71 |
| 15 | r | | 114 | 72 |
| 16 | s | | 115 | 73 |
| 17 | t | | 116 | 74 |
| 18 | u | | 117 | 75 |
| 19 | v | | 118 | 76 |
| 20 | w | | 119 | 77 |
| 21 | x | | 120 | 78 |
| 22 | y | | 121 | 79 |
| 23 | z | | 122 | 7A |
| 24 | { | Opening brace | 123 | 7B |
| 25 | ¦ | Split vertical bar | 124 | 7C |
| 26 | } | Closing brace | 125 | 7D |
| 27 | ~ | Tilde | 126 | 7E |

# Appendix C. Source Language Debugging

COBOL language elements that implement the debugging feature are:

- Debugging lines
- Debugging sections
- DEBUG-ITEM special register
- Compile-time switch (WITH DEBUGGING MODE clause)
- Object-time switch

## Coding Debugging Lines

A debugging line is a statement that is compiled only when the compile-time switch is activated. Debugging lines allow you, for example, to check the value of a data-name at certain points in a procedure.

To specify a debugging line in your program, code a “D” in column 7 (the indicator area). You can include successive debugging lines, but each must have a “D” in column 7 and you cannot break character strings across two lines.

All your debugging lines must be written so that the program is syntactically correct, whether the debugging lines are compiled or treated as comments.

You can code debugging lines anywhere in your program after the OBJECT-COMPUTER paragraph.

If a debugging line contains only spaces in Area A and in Area B, it is treated as a blank line.

## Coding Debugging Sections

Debugging sections are only permitted in the outermost program; they are not valid in nested programs. Debugging sections are never triggered by procedures contained in nested programs.

Debugging sections are declarative procedures. Declarative procedures are described under “USE Statement” on page 508. A debugging section can be invoked, for example, by a PERFORM statement that causes repeated execution of a procedure. Any associated procedure-name debugging declarative section is executed once for each repetition.

A debugging section executes *only* if both the compile-time switch and the object-time switch are activated.

The debug feature recognizes each separate occurrence of an imperative statement *within* an imperative statement as the beginning of a separate statement.

© Copyright IBM Corp. 1991, 1996

## Source Language Debugging

You cannot refer to a procedure defined within a debugging section in a statement outside of the debugging section.

References to the DEBUG-ITEM special register can be made only from within a debugging declarative procedure.

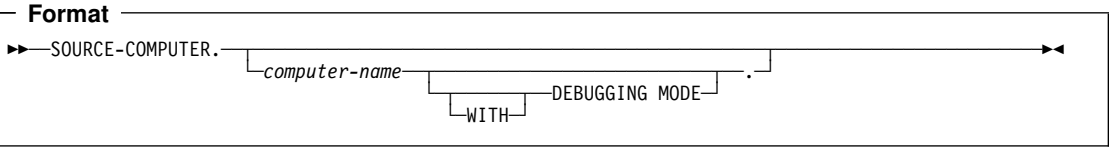
## DEBUG-ITEM Special Register

For information on the DEBUG-ITEM special register, see "DEBUG-ITEM Special Register."

## Activate Compile-Time Switch

The compile-time switch activates the debugging lines and sections. To place the compile-time switch in effect, specify WITH DEBUGGING MODE in the SOURCE COMPUTER paragraph of the Configuration Section.

**Format**




**WITH DEBUGGING MODE**

When WITH DEBUGGING MODE is specified, all debugging sections and debugging lines are compiled.

When WITH DEBUGGING MODE is omitted, all debugging sections and debugging lines are treated as comments.

**Note:** If you include a COPY statement as a debugging line, the "D" must appear on the first line of the COPY statement. IBM COBOL treats the copied text as the debugging line or lines. The COPY statement is executed, regardless of whether WITH DEBUGGING MODE is specified or not.

## Activate Object-Time Switch

The object-time switch is set when the run-time option DEBUG or NODEBUG is specified. (DEBUG is the default supplied by IBM.)

For details on the format, see:

- *Language Environment Programming Guide* for MVS and VM
- *IBM COBOL Set for AIX Programming Guide* for AIX
- *IBM VisualAge for COBOL for OS/2 Programming Guide* for OS/2 and Windows

The USE FOR DEBUGGING declarative procedures are activated when DEBUG is in effect and inhibited when NODEBUG is in effect.

The debugging lines (D in column 7) are not affected by the DEBUG/NODEBUG option; they are always active if they have been compiled.

When WITH DEBUGGING MODE is **not** specified in the SOURCE-COMPUTER paragraph, the object-time switch has no effect on execution of the object program.

You do not have to recompile the source program to activate or deactivate the object-time switch.

# Appendix D. Reserved Words

This list identifies all reserved words in the IBM COBOL for MVS & VM, IBM VisualAge for COBOL for OS/2, and IBM COBOL Set for AIX products. It also identifies words that are reserved in the COBOL 85 Standard (which are not reserved in the IBM COBOL products), and words reserved for future development.

- Words marked under **IBM COBOL** are reserved words in the IBM COBOL for MVS & VM, IBM COBOL Set for AIX, and IBM VisualAge for COBOL for OS/2 products. These reserved words include both reserved words for IBM extensions and a subset of the COBOL 85 Standard reserved words.

  Reserved words applicable to AIX, OS/2, and Windows only are identified with the following notation: [1].
- Words marked under **Standard Only** are COBOL 85 Standard reserved words that are **not** reserved in the IBM COBOL products. If used as user-defined names, these words are flagged with an S-LEVEL message.
- Words marked under **RFD** are reserved for future development and are flagged with an I-LEVEL message.

**Note:** You can change the contents of the reserved word table by using the WORD compiler option. For details, on how to specify an alternate reserved word table, see the *IBM COBOL Programming Guide* for your platform.

*Table 53 (Page 1 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| ACCEPT | X | | |
| ACCESS | X | | |
| ADD | X | | |
| ADDRESS | X | | |
| ADVANCING | X | | |
| AFTER | X | | |
| ALL | X | | |
| ALLOWING | | | X |
| ALPHABET | X | | |
| ALPHABETIC | X | | |
| ALPHABETIC-LOWER | X | | |
| ALPHABETIC-UPPER | X | | |
| ALPHANUMERIC | X | | |
| ALPHANUMERIC-EDITED | X | | |
| ALSO | X | | |
| ALTER | X | | |
| ALTERNATE | X | | |
| AND | X | | |
| ANY | X | | |
| APPLY | X | | |
| ARE | X | | |

*Table 53 (Page 1 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| AREA | X | | |
| AREAS | X | | |
| ARITHMETIC | | | X |
| ASCENDING | X | | |
| ASSIGN | X | | |
| AT | X | | |
| AUTHOR | X | | |
| AUTOMATIC | X[1] | | |
| B-AND | | | X |
| B-EXOR | | | X |
| B-LESS | | | X |
| B-NOT | | | X |
| B-OR | | | X |
| BASIS | X | | |
| BEFORE | X | | |
| BEGINNING | X | | |
| BINARY | X | | |
| BIT | | | X |
| BITS | | | X |
| BLANK | X | | |
| BLOCK | X | | |


© Copyright IBM Corp. 1991, 1996

*Table 53 (Page 2 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| BOOLEAN | | | X |
| BOTTOM | X | | |
| BY | X | | |
| CALL | X | | |
| CANCEL | X | | |
| CBL | X | | |
| CD | | X | |
| CF | | X | |
| CH | | X | |
| CHARACTER | X | | |
| CHARACTERS | X | | |
| CLASS | X | | |
| X CLASS-ID | X | | |
| CLOCK-UNITS | | X | |
| CLOSE | X | | |
| COBOL | X | | |
| CODE | X | | |
| CODE-SET | X | | |
| COLLATING | X | | |
| COLUMN | | X | |
| COM-REG | X | | |
| COMMA | X | | |
| COMMIT | | | X |
| COMMON | X | | |
| COMMUNICATION | | X | |
| COMP | X | | |
| COMP-1 | X | | |
| COMP-2 | X | | |
| COMP-3 | X | | |
| COMP-4 | X | | |
| \| COMP-5 | X[1] | | X[2] |
| COMP-6 | | | X |
| COMP-7 | | | X |
| COMP-8 | | | X |
| COMP-9 | | | X |
| COMPUTATIONAL | X | | |
| COMPUTATIONAL-1 | X | | |
| COMPUTATIONAL-2 | X | | |
| COMPUTATIONAL-3 | X | | |
| COMPUTATIONAL-4 | X | | |
| \| COMPUTATIONAL-5 | X[1] | | X[2] |
| COMPUTATIONAL-6 | | | X |
| COMPUTATIONAL-7 | | | X |
| COMPUTATIONAL-8 | | | X |
| COMPUTATIONAL-9 | | | X |

*Table 53 (Page 2 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| COMPUTE | X | | |
| CONFIGURATION | X | | |
| CONNECT | | | X |
| CONTAINED | | | X |
| CONTAINS | X | | |
| CONTENT | X | | |
| CONTINUE | X | | |
| CONTROL | | X | |
| CONTROLS | | X | |
| CONVERTING | X | | |
| COPY | X | | |
| CORR | X | | |
| CORRESPONDING | X | | |
| COUNT | X | | |
| CURRENCY | X | | |
| CURRENT | | | X |
| CYCLE | | | X |
| DATA | X | | |
| DATE | X | | |
| DATE-COMPILED | X | | |
| DATE-WRITTEN | X | | |
| DAY | X | | |
| DAY-OF-WEEK | X | | |
| DB | | | X |
| DB-ACCESS-CONTROL-KEY | | | X |
| DB-DATA-NAME | | | X |
| DB-EXCEPTION | | | X |
| DB-RECORD-NAME | | | X |
| DB-SET-NAME | | | X |
| DB-STATUS | | | X |
| DBCS | X | | |
| DE | | X | |
| DEBUG-CONTENTS | X | | |
| DEBUG-ITEM | X | | |
| DEBUG-LINE | X | | |
| DEBUG-NAME | X | | |
| DEBUG-SUB-1 | X | | |
| DEBUG-SUB-2 | X | | |
| DEBUG-SUB-3 | X | | |
| DEBUGGING | X | | |
| DECIMAL-POINT | X | | |
| DECLARATIVES | X | | |
| DEFAULT | | | X |
| DELETE | X | | |
| DELIMITED | X | | |

## Reserved Words

*Table 53 (Page 3 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| DELIMITER | X | | |
| DEPENDING | X | | |
| DESCENDING | X | | |
| DESTINATION | | X | |
| DETAIL | | X | |
| DISABLE | | | X |
| DISCONNECT | | | X |
| DISPLAY | X | | |
| DISPLAY-1 | X | | |
| DISPLAY-2 | | | X |
| DISPLAY-3 | | | X |
| DISPLAY-4 | | | X |
| DISPLAY-5 | | | X |
| DISPLAY-6 | | | X |
| DISPLAY-7 | | | X |
| DISPLAY-8 | | | X |
| DISPLAY-9 | | | X |
| DIVIDE | X | | |
| DIVISION | X | | |
| DOWN | X | | |
| DUPLICATE | | | X |
| DUPLICATES | X | | |
| DYNAMIC | X | | |
| EGCS | X | | |
| EGI | | X | |
| EJECT | X | | |
| ELSE | X | | |
| EMI | | X | |
| EMPTY | | | X |
| ENABLE | | X | |
| END | X | | |
| END-ADD | X | | |
| END-CALL | X | | |
| END-COMPUTE | X | | |
| END-DELETE | X | | |
| END-DISABLE | | | X |
| END-DIVIDE | X | | |
| END-ENABLE | | | X |
| END-EVALUATE | X | | |
| END-IF | X | | |
| END-INVOKE | X | | |
| END-MULTIPLY | X | | |
| END-OF-PAGE | X | | |
| END-PERFORM | X | | |
| END-READ | X | | |

*Table 53 (Page 3 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| END-RECEIVE | | X | |
| END-RETURN | X | | |
| END-REWRITE | X | | |
| END-SEARCH | X | | |
| END-SEND | | | X |
| END-START | X | | |
| END-STRING | X | | |
| END-SUBTRACT | X | | |
| END-TRANSCEIVE | | | X |
| END-UNSTRING | X | | |
| END-WRITE | X | | |
| ENDING | X | | |
| ENTER | X | | |
| ENTRY | X | | |
| ENVIRONMENT | X | | |
| EOP | X | | |
| EQUAL | X | | |
| EQUALS | | | X |
| ERASE | | | X |
| ERROR | X | | |
| ESI | | X | |
| EVALUATE | X | | |
| EVERY | X | | |
| EXACT | | | X |
| EXCEEDS | | | X |
| EXCEPTION | X | | |
| EXCLUSIVE | | | X |
| EXIT | X | | |
| EXTEND | X | | |
| EXTERNAL | X | | |
| FALSE | X | | |
| FD | X | | |
| FETCH | | | X |
| FILE | X | | |
| FILE-CONTROL | X | | |
| FILLER | X | | |
| FINAL | | X | |
| FIND | | | X |
| FINISH | | | X |
| FIRST | X | | |
| FOOTING | X | | |
| FOR | X | | |
| FORM | | | X |
| FORMAT | | | X |
| FREE | | | X |

*Table 53 (Page 4 of 7). Reserved Words*

| | Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|---|
| | FROM | X | | |
| | FUNCTION | X | | |
| | GENERATE | | X | |
| | GET | | | X |
| | GIVING | X | | |
| | GLOBAL | X | | |
| | GO | X | | |
| | GOBACK | X | | |
| | GREATER | X | | |
| | GROUP | | X | |
| | HEADING | | X | |
| | HIGH-VALUE | X | | |
| | HIGH-VALUES | X | | |
| | I-O | X | | |
| | I-O-CONTROL | X | | |
| | ID | X | | |
| | IDENTIFICATION | X | | |
| | IF | X | | |
| | IN | X | | |
| | INDEX | X | | |
| | INDEX-1 | | | X |
| | INDEX-2 | | | X |
| | INDEX-3 | | | X |
| | INDEX-4 | | | X |
| | INDEX-5 | | | X |
| | INDEX-6 | | | X |
| | INDEX-7 | | | X |
| | INDEX-8 | | | X |
| | INDEX-9 | | | X |
| | INDEXED | X | | |
| | INDICATE | | X | |
| X | INHERITS | X | | |
| | INITIAL | X | | |
| | INITIALIZE | X | | |
| | INITIATE | | X | |
| | INPUT | X | | |
| | INPUT-OUTPUT | X | | |
| | INSERT | X | | |
| | INSPECT | X | | |
| | INSTALLATION | X | | |
| | INTO | X | | |
| | INVALID | X | | |
| X | INVOKE | X | | |
| | IS | X | | |
| | JUST | X | | |

*Table 53 (Page 4 of 7). Reserved Words*

| | Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|---|
| | JUSTIFIED | X | | |
| | KANJI | X | | |
| | KEEP | | | X |
| | KEY | X | | |
| | LABEL | X | | |
| | LAST | | X | |
| | LD | | | X |
| | LEADING | X | | |
| | LEFT | X | | |
| | LENGTH | X | | |
| | LESS | X | | |
| | LIMIT | | X | |
| | LIMITS | | X | |
| | LINAGE | X | | |
| | LINAGE-COUNTER | X | | |
| | LINE | X | | |
| | LINE-COUNTER | | X | |
| | LINES | X | | |
| | LINKAGE | X | | |
| | LOCALLY | | | X |
| X | LOCAL-STORAGE | X | | |
| | LOCK | X | | |
| | LOW-VALUE | X | | |
| | LOW-VALUES | X | | |
| | MEMBER | | | X |
| | MEMORY | X | | |
| | MERGE | X | | |
| | MESSAGE | | X | |
| X | METACLASS | X | | |
| X | METHOD | X | | |
| X | METHOD-ID | X | | |
| | MODE | X | | |
| | MODIFY | | | X |
| | MODULES | X | | |
| | MORE-LABELS | X | | |
| | MOVE | X | | |
| | MULTIPLE | X | | |
| | MULTIPLY | X | | |
| | NATIVE | X | | |
| | NEGATIVE | X | | |
| | NEXT | X | | |
| | NO | X | | |
| | NORMAL | | | X |
| | NOT | X | | |
| | NULL | X | | |

## Reserved Words

*Table 53 (Page 5 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| NULLS | X | | |
| NUMBER | | X | |
| NUMERIC | X | | |
| NUMERIC-EDITED | X | | |
| OBJECT | X | | |
| OBJECT-COMPUTER | X | | |
| OCCURS | X | | |
| OF | X | | |
| OFF | X | | |
| OMITTED | X | | |
| ON | X | | |
| ONLY | | | X |
| OPEN | X | | |
| OPTIONAL | X | | |
| OR | X | | |
| ORDER | X | | |
| ORGANIZATION | X | | |
| OTHER | X | | |
| OUTPUT | X | | |
| OVERFLOW | X | | |
| OVERRIDE | X | | |
| OWNER | | | X |
| PACKED-DECIMAL | X | | |
| PADDING | X | | |
| PAGE | X | | |
| PAGE-COUNTER | | X | |
| PARAGRAPH | | | X |
| PASSWORD | X | | |
| PERFORM | X | | |
| PF | | X | |
| PH | | X | |
| PIC | X | | |
| PICTURE | X | | |
| PLUS | | X | |
| POINTER | X | | |
| POSITION | X | | |
| POSITIVE | X | | |
| PRESENT | | | X |
| PREVIOUS | X[1] | | X[2] |
| PRINTING | | X | |
| PRIOR | | | X |
| PROCEDURE | X | | |
| PROCEDURE-POINTER | X | | |
| PROCEDURES | X | | |
| PROCEED | X | | |

*Table 53 (Page 5 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| PROCESSING | X | | |
| PROGRAM | X | | |
| PROGRAM-ID | X | | |
| PROTECTED | | | X |
| PURGE | | X | |
| QUEUE | | X | |
| QUOTE | X | | |
| QUOTES | X | | |
| RANDOM | X | | |
| RD | | X | |
| READ | X | | |
| READY | X | | |
| REALM | | | X |
| RECEIVE | | X | |
| RECONNECT | | | X |
| RECORD | X | | |
| RECORD-NAME | | | X |
| RECORDING | X | | |
| RECORDS | X | | |
| RECURSIVE | X | | |
| REDEFINES | X | | |
| REEL | X | | |
| REFERENCE | X | | |
| REFERENCES | X | | |
| RELATION | | | X |
| RELATIVE | X | | |
| RELEASE | X | | |
| RELOAD | X | | |
| REMAINDER | X | | |
| REMOVAL | X | | |
| RENAMES | X | | |
| REPEATED | | | X |
| REPLACE | X | | |
| REPLACING | X | | |
| REPORT | | X | |
| REPORTING | | X | |
| REPORTS | | X | |
| REPOSITORY | X | | |
| RERUN | X | | |
| RESERVE | X | | |
| RESET | X | | |
| RETAINING | | | X |
| RETRIEVAL | | | X |
| RETURN | X | | |
| RETURN-CODE | X | | |

*Table 53 (Page 6 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| RETURNING | X | | |
| REVERSED | X | | |
| REWIND | X | | |
| REWRITE | X | | |
| RF | | X | |
| RH | | X | |
| RIGHT | X | | |
| ROLLBACK | | | X |
| ROUNDED | X | | |
| RUN | X | | |
| SAME | X | | |
| SD | X | | |
| SEARCH | X | | |
| SECTION | X | | |
| SECURITY | X | | |
| SEGMENT | | X | |
| SEGMENT-LIMIT | X | | |
| SELECT | X | | |
| SELF | X | | |
| SEND | | X | |
| SENTENCE | X | | |
| SEPARATE | X | | |
| SEQUENCE | X | | |
| SEQUENTIAL | X | | |
| SERVICE | X | | |
| SESSION-ID | | | X |
| SET | X | | |
| SHARED | | | X |
| SHIFT-IN | X | | |
| SHIFT-OUT | X | | |
| SIGN | X | | |
| SIZE | X | | |
| SKIP1 | X | | |
| SKIP2 | X | | |
| SKIP3 | X | | |
| SORT | X | | |
| SORT-CONTROL | X | | |
| SORT-CORE-SIZE | X | | |
| SORT-FILE-SIZE | X | | |
| SORT-MERGE | X | | |
| SORT-MESSAGE | X | | |
| SORT-MODE-SIZE | X | | |
| SORT-RETURN | X | | |
| SOURCE | | X | |
| SOURCE-COMPUTER | X | | |

*Table 53 (Page 6 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| SPACE | X | | |
| SPACES | X | | |
| SPECIAL-NAMES | X | | |
| STANDARD | X | | |
| STANDARD-1 | X | | |
| STANDARD-2 | X | | |
| STANDARD-3 | | | X |
| STANDARD-4 | | | X |
| START | X | | |
| STATUS | X | | |
| STOP | X | | |
| STORE | | | X |
| STRING | X | | |
| SUB-QUEUE-1 | | X | |
| SUB-QUEUE-2 | | X | |
| SUB-QUEUE-3 | | X | |
| SUB-SCHEMA | | | X |
| SUBTRACT | X | | |
| SUM | X | | |
| SUPER | X | | |
| SUPPRESS | X | | |
| SYMBOLIC | X | | |
| SYNC | X | | |
| SYNCHRONIZED | X | | |
| TABLE | | X | |
| TALLY | X | | |
| TALLYING | X | | |
| TAPE | X | | |
| TENANT | | | X |
| TERMINAL | | X | |
| TERMINATE | | X | |
| TEST | X | | |
| TEXT | | X | |
| THAN | X | | |
| THEN | X | | |
| THROUGH | X | | |
| THRU | X | | |
| TIME | X | | |
| TIMEOUT | | | X |
| TIMES | X | | |
| TITLE | X | | |
| TO | X | | |
| TOP | X | | |
| TRACE | X | | |
| TRAILING | X | | |

## Reserved Words

*Table 53 (Page 7 of 7). Reserved Words*

| Reserved Word | IBM COBOL | Standard Only | RFD |
|---|---|---|---|
| TRANSCEIVE | | | X |
| TRUE | X | | |
| TYPE | | X | |
| UNEQUAL | | | X |
| UNIT | X | | |
| UNSTRING | X | | |
| UNTIL | X | | |
| UP | X | | |
| UPDATE | | | X |
| UPON | X | | |
| USAGE | X | | |
| USAGE-MODE | | | X |
| USE | X | | |
| USING | X | | |
| VALID | | | X |
| VALIDATE | | | X |
| VALUE | X | | |
| VALUES | X | | |
| VARYING | X | | |
| WAIT | | | X |
| WHEN | X | | |
| WHEN-COMPILED | X | | |
| WITH | X | | |
| WITHIN | | | X |
| WORDS | X | | |
| WORKING-STORAGE | X | | |
| WRITE | X | | |
| WRITE-ONLY | X | | |
| ZERO | X | | |
| ZEROES | X | | |
| ZEROS | X | | |
| < | X | | |
| <= | X | | |
| + | X | | |
| * | X | | |
| ** | X | | |
| - | X | | |
| / | X | | |
| > | X | | |
| >= | X | | |
| = | X | | |

**Note:** 1 These words are reserved under AIX, OS/2, and Windows only.

2 These words are reserved for future development (RFD) under MVS and VM only.

# Appendix E. ASCII Considerations for MVS and VM

The compiler supports the American National Standard Code for Information Interchange (ASCII). Thus, the programmer can create and process tape files recorded in accordance with the following standards:

- American National Standard Code for Information Interchange, X3.4-1977
- American National Standard Magnetic Tape Labels for Information Interchange, X3.27-1978
- American National Standard Recorded Magnetic Tape for Information Interchange (800 CPI, NRZI), X3.22-1967.

ASCII-encoded tape files, when read into the system, are automatically translated in the buffers into EBCDIC. Internal manipulation of data is performed exactly as if the ASCII files were EBCDIC-encoded files. For an output file, the system translates the EBCDIC characters into ASCII in the buffers before writing the file on tape. Therefore, there are special considerations concerning ASCII-encoded files when they are processed in COBOL.

This appendix also applies (with appropriate modifications) to the International Reference Version of the ISO 7-bit code (ISCII) defined in International Standard 646, 7-Bit Coded Character Set for Information Processing Interchange. The ISCII code set differs from ASCII only in the graphic representation of two code points:

- Ordinal number 37, which is a dollar sign in ASCII, but a lozenge in ISCII
- Ordinal number 127, which is a tilde (~) in ASCII, but an overline (or optionally a tilde) in ISCII.

**Note:** In the following discussion, the information given for STANDARD-1 also applies to STANDARD-2 except where otherwise specified.

The following paragraphs discuss the special considerations concerning ASCII- (or ISCII-) encoded files.

## Environment Division

In the Environment Division, the OBJECT-COMPUTER, SPECIAL-NAMES, and FILE-CONTROL paragraphs are affected.

### OBJECT-COMPUTER and SPECIAL-NAMES Paragraphs

When at least one file in the program is an ASCII-encoded file, the alphabet-name clause of the SPECIAL-NAMES paragraph must be specified; the alphabet-name must be associated with STANDARD-1 or STANDARD-2 (for ASCII or ISCII collating sequence or CODE SET, respectively).

© Copyright IBM Corp. 1991, 1996

## ASCII Considerations

When nonnumeric comparisons within the object program are to use the ASCII collating sequence, the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph must be specified; the alphabet-name used must also be specified as an alphabet-name in the SPECIAL-NAMES paragraph, and associated with STANDARD-1. For example:

```
Object-computer.  IBM-370
    Program collating sequence is ASCII-sequence.
Special-names.  Alphabet ASCII-sequence is standard-1.
```

When both clauses are specified, the ASCII collating sequence is used in this program to determine the truth value of the following nonnumeric comparisons:

- Those explicitly specified in relation conditions
- Those explicitly specified in condition-name conditions
- Any nonnumeric sort or merge keys (unless the COLLATING SEQUENCE phrase is specified in the MERGE or SORT statement).

When the PROGRAM COLLATING SEQUENCE clause is omitted, the EBCDIC collating sequence is used for such comparisons.

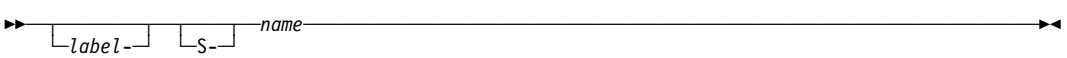
The PROGRAM COLLATING SEQUENCE clause, in conjunction with the alphabet-name clause, can be used to specify EBCDIC nonnumeric comparisons for an ASCII-encoded tape file or ASCII nonnumeric comparisons for an EBCDIC-encoded tape file.

The literal option of the alphabet-name clause can be used to process internal data in a collating sequence other than NATIVE or STANDARD-1.

## FILE-CONTROL Paragraph

For ASCII files, the ASSIGN clause assignment-name has the following formats:

**Format—QSAM File**




The file must be a QSAM file assigned to a magnetic tape device.

**label-**
Documents the device and device class to which a file is assigned. If specified, it must end with a hyphen.

**S-** The organization field. Optional for QSAM files, which always have sequential organization.

**name**
A required 1- to 8-character field that specified the external name for this file.

## I-O-CONTROL Paragraph

The assignment-name in a RERUN clause must not specify an ASCII-encoded file.

ASCII-encoded files containing checkpoint records cannot be processed.

## Data Division

In the Data Division, there are special considerations for the FD entry and for data description entries.

For each logical file defined in the Environment Division, there must be a corresponding FD entry and level-01 record description entry in the File Section of the Data Division.

## FD Entry—CODE-SET Clause

The FD Entry for an ASCII-encoded file must contain a CODE-SET clause; the alphabet-name must be associated with STANDARD-1 (for the ASCII code set) in the SPECIAL-NAMES paragraph. For example:

```
Special-names.  Alphabet ASCII-sequence is standard-1.
   .
   .
   .
FD  ASCII-file label records standard
  Recording mode is f
  Code-set is ASCII-sequence.
```

## Data Description Entries

For ASCII files, the following data description considerations apply:

- PICTURE clause specifications for all five categories of data are valid.
- For signed numeric items, the SIGN clause with the SEPARATE CHARACTER phrase must be specified.
- For the USAGE clause, only the DISPLAY phrase is valid.

## Procedure Division

An ASCII collated sort/merge operation can be specified in two ways:

- Through the PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph.

  In this case, the ASCII collating sequence is used for nonnumeric comparisons explicitly specified in relation conditions and condition-name conditions.

- Through the COLLATING SEQUENCE phrase of the SORT or MERGE statement.

  In this case, only this sort/merge operation uses the ASCII collating sequence.

In either case, alphabet-name must be associated with STANDARD-1 (for ASCII collating sequence) in the SPECIAL-NAMES paragraph.

## ASCII Considerations

For this sort/merge operation, the COLLATING SEQUENCE option takes precedence over the PROGRAM COLLATING SEQUENCE clause.

If both the PROGRAM COLLATING SEQUENCE clause and the COLLATING SEQUENCE phrase are omitted (or if the one in effect specifies an EBCDIC collating sequence), the sort/merge is performed using the EBCDIC collating sequence.

# Appendix F. Locale Considerations (Workstation Only)

A *locale* is defined by language-specific and cultural-specific conventions for processing information. All such information should be accessible to a program at run time so that the same program can display or process data differently for different countries.

Locale information consists of data from six categories. Each locale is described by a locale definition file. The following standard categories can be defined in a locale definition source file:

**LC_CTYPE** Defines character classification, case conversion, and other character attributes. Use this category to define the code page in effect.

**LC_COLLATE** Defines string-collation order information. For IBM COBOL Set for AIX and IBM VisualAge for COBOL for OS/2, this defines the collating sequence in effect. This only impacts any > or < comparisons, such as relational conditions and the SORT or MERGE verb.

**LC_MESSAGES** Defines the format for affirmative and negative responses. For IBM COBOL Set for AIX and IBM VisualAge for COBOL for OS/2, this impacts whether messages (error messages and listing headers for example) are in US English or Japanese. For any locale other than Japanese, US English is used.

**LC_MONETARY** Defines rules an symbols for formatting monetary numeric information.

For IBM COBOL Set for AIX and IBM VisualAge for COBOL for OS/2 this attribute has no affect. Monetary value representation is controlled through the COBOL language syntax.

**LC_NUMERIC** Defines rules and symbols for formatting nonmonetary numeric information.

For IBM COBOL Set for AIX and IBM VisualAge for COBOL for OS/2 this attribute has no affect. Nonmonetary numeric value representation is controlled through the COBOL language syntax.

**LC_TIME** Lists rules and symbols for formatting time and date information.

For IBM COBOL Set for AIX and IBM VisualAge for COBOL for OS/2 this attribute only affects the date and time shown on the compiler listings. All other date and time values are controlled through the COBOL language syntax.

Locale definition files are named by the language, territory, and code set information they describe.

© Copyright IBM Corp. 1991, 1996

# Appendix G. Summary of Language Difference: Host COBOL and Workstation COBOL

Table 54 table lists the language elements that are different between IBM COBOL for MVS & VM and the IBM Workstation COBOL compilers Many IBM COBOL for MVS & VM clauses and phrases are treated as comments under AIX, OS/2, and Windows. However, this will have minimal effect on existing applications that you download to the workstation. The Workstation compilers recognize and process most IBM COBOL for MVS & VM language syntax, even if the language element has no functional effect.

*Table 54 (Page 1 of 2). Language Difference Between Mainframe and Workstation IBM COBOL*

| Language Element | Implementation |
|---|---|
| ACCEPT statement | Under AIX, OS/2, and Windows, environment-name determines file identification. |
| APPLY WRITE-ONLY clause | Treated as a comment under AIX, OS/2, and Windows. |
| ASSIGN clause | Different syntax for the ASSIGNment name. ASSIGN...USING data-name is not supported on MVS and VM. |
| BLOCK CONTAINS clause | Treated as a comment under AIX, OS/2, and Windows. |
| CALL statement | A file-name as a CALL argument is not supported under AIX, OS/2, and Windows. |
| CLOSE statement | The following phrases are treated as comments under AIX, OS/2, and Windows: FOR REMOVAL, WITH NO REWIND, and UNIT/REEL. |
| CODE-SET clause | Treated as a comment under AIX, OS/2, and Windows. |
| DISPLAY statement | Under AIX, OS/2, and Windows, environment-name determines file identification. |
| File Status data-name-1 | Some values and meanings for file status 9x are different under MVS and VM than under AIX, OS/2, and Windows. |
| File Status data-name-8 | The format and values are different depending on the platform and the file system. |
| LABEL RECORD clause | LABEL RECORD IS *data-name*, USE...AFTER...LABEL PROCEDURE, and GO TO MORE-LABELS are treated as comments under AIX, OS/2, and Windows. These language elements are processed by the compiler; however, the user label declaratives are not invoked at run time. |
| Line Sequential files | Addition of line sequential I-O for AIX, OS/2, and Windows. |
| LOCK MODE clause | Under AIX, OS/2, and Windows only, you can use this clause to specify the locking technique used for files. |
| MULTIPLE FILE TAPE | Treated as a comment under AIX, OS/2, and Windows. On the workstation, all files are treated as single volume files. |
| OPEN statement | The following phrases are treated as comments under AIX, OS/2, and Windows: REVERSED and WITH NO REWIND. |
| PASSWORD clause | Treated as a comment under AIX, OS/2, and Windows. |


© Copyright IBM Corp. 1991, 1996

*Table 54 (Page 2 of 2). Language Difference Between Mainframe and Workstation IBM COBOL*

| Language Element | Implementation |
|---|---|
| POINTER and PROCEDURE-POINTER data items | Under IBM COBOL for MVS & VM, a POINTER data item is defined as 4 bytes; a PROCEDURE-POINTER data item is defined as 8 bytes. Under AIX or OS/2, the size of these data items are consistent with the native pointer definition of the platform (4 bytes for 32-bit machines and 8 bytes for 64-bit machines). |
| READ...PREVIOUS | Under AIX, OS/2, and Windows only, allows you to read the previous record for relative or index files with DYNAMIC access mode. |
| RECORD CONTAINS clause | The RECORD CONTAINS n CHARACTERS clause is accepted with one exception: RECORD CONTAINS 0 CHARACTERS is treated as a comment under AIX, OS/2, and Windows. |
| RECORDING MODE clause | Treated as a comment under AIX, OS/2, and Windows for relative, indexed, and line sequential files. |
| RERUN clause | Treated as a comment under AIX, OS/2, and Windows. |
| RESERVE clause | Treated as a comment under AIX, OS/2, and Windows. |
| SAME AREA clause | Treated as a comment under AIX, OS/2, and Windows. |
| SAME SORT clause | Treated as a comment under AIX, OS/2, and Windows. |
| SET statement | Under AIX, OS/2, and Windows, format 5, rules for *identifier-7* have extensions. |
| SORT-CONTROL special register | Has a different implicit definition. |
| SORT-CORE-SIZE special register | The contents of this special register differ between host and workstation COBOL. |
| SORT-FILE-SIZE special register | Treated as a comment under AIX, OS/2, and Windows. Values in this special register are not used. |
| SORT-MESSAGE special register | Treated as a comment under AIX, OS/2, and Windows. |
| SORT-MODE-SIZE special register | Treated as a comment under AIX, OS/2, and Windows. Values in this special register are not used. |
| SORT MERGE AREA clause | Treated as a comment under AIX, OS/2, and Windows. |
| START...< | Under AIX, OS/2, and Windows, the following relational operators are allowed: IS LESS THAN, IS <, IS NOT GREATER THAN, IS NOT >, IS LESS THAN OR EQUAL TO, IS <= |
| USAGE clause | Under AIX, OS/2, and Windows, addition of COMP-5 (new data type, native binary data format). |
| WRITE statement | Under AIX, OS/2, and Windows, if you specify the WRITE...ADVANCING with environment names: C01 - C12 or S01 - S05, one line is advanced. |
| Names known to the platform environment | The following names are identified differently: program-name, text-name, library-name, assignment-name, SORT-CONTROL special register, basis-name, DISPLAY/ACCEPT target identification, and system-dependent names. |

# Appendix H. Industry Specifications

The following industry specifications are supported:

1. ISO 1989:1985, Programming languages - COBOL.

   ISO 1989/Amendment 1, Programming languages - COBOL - Amendment 1: Intrinsic function module.

   ISO 1989:1985 is identical to X3.23-1985, American National Standard for Information Systems - Programming Language - COBOL.

   ISO 1989/Amendment 1 is identical to X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL.

   ISO 1989:1985/Amd.2:1994, Programming languages - COBOL - Amendment 2: Correction and clarification amendment for COBOL

   ISO 1989:1985/Amd.2:1994 is identical to ANSI X3.23b-1993, American National Standard for Information Systems - Programming Language - Correction Amendment for COBOL

   For supported modules, see item 2 below.

2. X3.23-1985, American National Standard for Information Systems - Programming Language - COBOL.

   X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL. ANSI X3.23b-1993, American National Standard for Information Systems - Programming Language - Correction Amendment for COBOL

   All required modules are supported at the highest level defined by the standard. In the following list, the shorthand notation for describing module levels is shown in parentheses. For example, to summarize module information for sequential input/output, the shorthand notation is (2 SEQ 1,2). The first digit indicates the level of language elements within the module supported by COBOL for MVS & VM. Next is the three-character abbreviation of the module name as used in the standard. Finally, the two digits separated by a comma indicate the minimum and maximum levels of the module. For example, (2 SEQ 1,2) means that COBOL for MVS & VM supports the sequential I-O module at level 2, while the range of levels in the module is from 1 (minimum) to 2 (maximum).

   - Nucleus (2 NUC 1,2)

     Provides internal processing of data within the four basic divisions of a program and the capability for defining and accessing tables.

   - Sequential I-O (2 SEQ 1,2)

     Provides access to records of a file in established sequence. The sequence is established as a result of writing the records to the file.

   - Relative I-O (2 REL 0,2)


© Copyright IBM Corp. 1991, 1996

Provides access to records in either a random or sequential manner. Each record is uniquely identified by an integer specifying the record's logical position in a file.

- Indexed I-O (2 INX 0,2)

  Provides access to records in either a random or sequential manner. Each record in an indexed file is uniquely identified by the value of a key within that record.

- Sort-Merge (1 SRT 0,1)

  Orders one or more files of records, or combines two or more identically ordered files of records, according to a set of user-specified keys.

- Inter-Program Communication (2 IPC 1,2)

  Allows a COBOL program to communicate with other programs through transfers of control and access to common data items.

- Source Text Manipulation (2 STM 0,2)

  Allows the insertion of source program text as part of the compilation of the source program. COBOL libraries contain texts which are available to the compiler at compile time and which can be treated by the compiler as part of the source program.

In addition, the following optional modules of the standard are supported:

- Intrinsic Functions (1 ITR 0,1)

  Provides the capability to reference a data item whose value is derived automatically at the time of reference during the execution of the object program.

- Debug (1 DEB 0,2)

  Monitors object program execution through declarative procedures, special debugging lines, and a special register, DEBUG-ITEM, which gives specific information about execution status.

- Segmentation (2 SEG 0,2)

  Refreshes independent segments when required.

The following optional modules of the standard are not supported:

- Report Writer
- Communications
- Debug (2 DEB 0,2)

3. MVS/VM FIPS Publication 21-4, Federal Information Processing Standard 21-4, COBOL high subset.
4. International Reference Version of the ISO 7-bit code defined in *International Standard 646, 7-Bit Coded Character Set for Information Processing Interchange*.
5. The 7-bit coded character sets defined in *American National Standard X3.4-1977, Code for Information Interchange*.

## Industry Specifications

6. WRKSTN SPIRIT (Service Provider's Requirements for Information Technology), Part 6—COBOL Language Profile, published by Network Management Forum.
7. WRKSTN MIA (Multivendor Integration Architecture), technical requirements, specified by Nippon Telegraph and Telephone Corp (NTT).

## Standard Terminology

The term "COBOL 85 Standard" is used in this book to refer to the combination of the following standards:

1. ISO 1989:1985, Programming languages - COBOL.

   ISO 1989/Amendment 1, Programming languages - COBOL - Amendment 1: Intrinsic function module.

2. X3.23-1985, American National Standard for Information Systems - Programming Language - COBOL.

   X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL.

**Note:** The term "COBOL 74 Standard" is used in this book to refer to X3.23-1974, American National Standard for Information Systems - Programming Language - COBOL.

# Bibliography

## IBM COBOL for MVS & VM

*Fact Sheet*, GC26-8664

*Licensed Program Specifications*, GC26-4761.

*Compiler and Run-Time Migration Guide*, GC26-4764

*Installation and Customization under MVS*, GC26-4766

*Programming Guide*, SC26-4767

*Language Reference*, SC26-4769

*Debug Tool User's Guide and Reference*, SC09-2137

*Diagnosis Guide*, SC26-3138

### *Related Publications for MVS & VM*

- **IBM Language Environment for MVS & VM**

  *Fact Sheet*, GC26-4785

  *Concepts Guide*, GC26-4786

  *Installation and Customization*, SC26-4817

  *Programming Guide*, SC26-4818

  *Programming Reference*, SC26-3312

  *Writing Interlanguage Communications Applications*, SC26-8351

  *Run-Time Migration Guide*, SC26-8232

  *Debugging Guide and Run-Time Messages*, SC26-4829

  *Diagnosis Guide*, SC26-3154

  *Licensed Program Specifications*, GC26-4774

  *Master Index*, SC26-3427

- **SOMobjects for MVS**

  *Introducing SOMobjects on MVS*, GC28-1529

  *Licensed Program Specifications*, GC28-1534

  *User's Guide*, GC28-1545

  *Class Library Reference*, SC28-1546

  *Reference Summary*, SC28-1547

- **Other Products**

  *DFSORT Application Programming Guide*, SC33-4035

  *DFSMS/MVS Macro Instruction for Datasets*, SC26-4913

  *IBM VS COBOL for OS/VS*, GC26-3857

  *IMS/VS Batch Termination Simulator: Program Reference and Operations*, SH20-5523

## IBM COBOL Set for AIX (SBOF-7334)

*Fact Sheet*, GC26-8484

*Language Reference*, SC26-4769

*Programming Guide*, SC26-8423

*Program Builder User's Guide*, SC09-2201

*LPEX User's Guide and Reference*, SC09-2202

*Getting Started*, GC26-8425 (not included in BOF)

### *SMARTdata UTILITIES for AIX (SBOF-6132)*

*VSAM in a Distributed Environment*, SC26-7064

*Data Description and Conversion*, SC26-7066

*Data Description and Conversion A Data Language Reference*, SC26-7092

*SMARTsort for OS/2 and AIX*, SC26-7099

### *Related Publications for AIX*

- **DB2**

  *DATABASE 2 AIX/6000 Command Reference*, SC09-1575

  *DATABASE 2 AIX/6000 Command Reference for Common Servers*, S20H-4645

  *Application Programming Guide*, S20H-4643

  *SQL Reference*, S20H-4665

- **CICS**

  *Application Programming Guide*, SC33-1568

  *Application Programming Reference*, SC33-1569

  *Administration Guide*, SC33-1562

  *Administration Reference*, SC33-1563

  *Encina for CICS*, SC33-1567

  *Planning and Installation Guide*, GC33-1561

  *Intercommunication Guide*, SC33-1564

  *Messages and Codes*, SC33-1566


© Copyright IBM Corp. 1991, 1996

*Problem Determination Guide*, SC33-1565

- **SOMobjects for AIX**

  *SOMobjects Base Toolkit User's Guide*, SC23-2680-01

  *SOMobjects Base Toolkit Programmer's Reference Manual*, SC23-2681-01

  *SOMobjects Base Toolkit Quick Reference Guide*, SC23-2682-01

- Other

  *Structured File Server Programmer's Guide and Reference for AIX, Encina*, SC33-1610-00.

## IBM VisualAge for COBOL for OS/2 (SBOF-7333)

*Fact Sheet*, GC26-8663

*Language Reference*, SC26-4769

*Programming Guide*, SC26-8419

*Getting Started*, GC26-8421 (not included in BOF)

*Building VisualAge for COBOL Parts for Fun and Profit*, SC26-8789-00

### *SMARTdata UTILITIES for OS/2 (SBOF-6131)*

*VSAM in a Distributed Environment*, SC26-7063

*Data Description and Conversion*, SC26-7091

*Data Description and Conversion A Data Language Reference*, SC26-7092

*SMARTsort for OS/2 and AIX*, SC26-7099

### *Related Publications for OS/2*

- **DB2**

  *Application Programming Guide*, S20H-4643

  *SQL Reference*, S20H-4665

- **CICS**

  *Installation*, GC33-1580

  *Customization*, SC33-1581

  *Operation*, SC33-1582

  *Reference Summary*, SX33-6100

  *Intercommunication*, SC33-1583

  *Problem Determination*, SC33-1584

  *Application Programming*, SC33-1585

  *Messages & Codes*, SC33-1586

- **SOMobjects for OS/2**

  *SOMobjects Developers Toolkit User Guide Version 2.1*

  *SOMobjects Developers Toolkit Programmers Reference Manual Version 2.1*

- **Other**

  *Btrieve Programmer's Manual*

# Glossary

The terms in this glossary are defined in accordance with their meaning in COBOL. These terms may or may not have the same meaning in other languages.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the following publications:

- *American National Standard Programming Language COBOL, ANSI X3.23-1985* (Copyright 1985 American National Standards Institute, Inc.), which was prepared by Technical Committee X3J4, which had the task of revising American National Standard COBOL, X3.23-1974.
- *American National Dictionary for Information Processing Systems* (Copyright 1982 by the Computer and Business Equipment Manufacturers Association).

American National Standard definitions are preceded by an asterisk (*).

## A

*** abbreviated combined relation condition**. The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

**abend**. Abnormal termination of program.

*** access mode**. The manner in which records are to be operated upon within a file.

*** actual decimal point**. The physical representation, using the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

*** alphabet-name**. A user-defined word, in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION, that assigns a name to a specific character set and/or collating sequence.

*** alphabetic character**. A letter or a space character.

*** alphanumeric character**. Any character in the computer's character set.

**alphanumeric-edited character**. A character within an alphanumeric character-string that contains at least one B, 0 (zero), or / (slash).

*** alphanumeric function**. A function whose value is composed of a string of one or more characters from the computer's character set.

*** alternate record key**. A key, other than the prime record key, whose contents identify a record within an indexed file.

**ANSI (American National Standards Institute)**. An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

*** argument**. An identifier, a literal, an arithmetic expression, or a function-identifier that specifies a value to be used in the evaluation of a function.

*** arithmetic expression**. An identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

*** arithmetic operation**. The process caused by the execution of an arithmetic statement, or the evaluation of an arithmetic expression, that results in a mathematically correct solution to the arguments presented.

*** arithmetic operator**. A single character, or a fixed two-character combination that belongs to the following set:

| Character | Meaning |
|---|---|
| + | addition |
| – | subtraction |
| * | multiplication |
| / | division |
| ** | exponentiation |

*** arithmetic statement**. A statement that causes an arithmetic operation to be executed. The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements.

**array**. In Language Environment, an aggregate consisting of data objects, each of which may be uniquely


© Copyright IBM Corp. 1991, 1996

referenced by subscripting. Roughly analogous to a COBOL table.

*** ascending key**. A key upon the values of which data is ordered, starting with the lowest value of the key up to the highest value of the key, in accordance with the rules for comparing data items.

**ASCII**. American National Standard Code for Information Interchange. The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange between data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

**Extension:** IBM has defined an extension to ASCII code (characters 128-255).

**assignment-name**. A name that identifies the organization of a COBOL file and the name by which it is known to the system.

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

*** AT END condition**. A condition caused:

1. During the execution of a READ statement for a sequentially accessed file, when no next logical record exists in the file, or 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 input file is not present.
2. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.
3. During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

# B

**big-endian**. Default format used by the mainframe and the AIX workstation to store binary data. In this format, the least significant digit is on the highest address. Compare with “little-endian.”

**binary item**. A numeric data item represented in binary notation (on the base 2 numbering system). Binary items have a decimal equivalent consisting of the decimal digits 0 through 9, plus an operational sign. The leftmost bit of the item is the operational sign.

**binary search**. A dichotomizing search in which, at each step of the search, the set of data elements is divided by two; some appropriate action is taken in the case of an odd number.

*** block**. A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with physical record.

**breakpoint**. A place in a computer program, usually specified by an instruction, where its execution may be interrupted by external intervention or by a monitor program.

**Btrieve**. A key-indexed record management system that allows applications to manage records by key value, sequential access method, or random access method. IBM COBOL supports COBOL sequential and indexed file I-O language through Btrieve.

**buffer**. A portion of storage used to hold input or output data temporarily.

**built-in function**. See “intrinsic function”.

**byte**. A string consisting of a certain number of bits, usually eight, treated as a unit, and representing a character.

# C

**callable services**. In Language Environment, a set of services that can be invoked by a COBOL program using the conventional Language Environment-defined call interface, and usable by all programs sharing the Language Environment conventions.

**called program**. A program that is the object of a CALL statement.

*** calling program**. A program that executes a CALL to another program.

**case structure**. A program processing logic in which a series of conditions is tested in order to make a choice between a number of resulting actions.

**cataloged procedure**. A set of job control statements placed in a partitioned data set called the procedure library (SYS1.PROCLIB). You can use cataloged procedures to save time and reduce errors coding JCL.

**century window**. The 100-year interval in which Language Environment assumes all 2-digit years lie. The Language Environment default century window begins 80 years before the system date.

*** character**. The basic indivisible unit of the language.

**character position**. The amount of physical storage required to store a single standard data format character described as USAGE IS DISPLAY.

**character set**. All the valid characters for a programming language or a computer system.

*** character-string**. A sequence of contiguous characters that form a COBOL word, a literal, a PICTURE character-string, or a comment-entry. Must be delimited by separators.

**checkpoint**. A point at which information about the status of a job and the system can be recorded so that the job step can be later restarted.

*** class**. The entity that defines common behavior and implementation for zero, one, or more objects. The objects that share the same implementation are considered to be objects of the same class.

*** class condition**. The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic, is wholly numeric, or consists exclusively of those characters listed in the definition of a class-name.

*** Class Definition**. The COBOL source unit that defines a class.

*** class identification entry**. An entry in the CLASS-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the class-name and assign selected attributes to the class definition.

*** class-name**. A user-defined word defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION that assigns a name to the proposition for which a truth value can be defined, that the content of a data item consists exclusively of those characters listed in the definition of the class-name.

**class object**. The run-time object representing a SOM class.

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

**CMS (Conversational Monitor System)**. A virtual machine operating system that provides general interactive, time-sharing, problem solving, and program development capabilities, and that operates only under the control of the VM/SP control program.

*** COBOL character set**. The complete COBOL character set consists of the characters listed below:

| Character | Meaning |
|---|---|
| 0,1...,9 | digit |
| A,B,...,Z | uppercase letter |
| a,b,...,z | lowercase letter |
| b | space |
| + | plus sign |
| – | minus sign (hyphen) |
| * | asterisk |
| / | slant (virgule, slash) |
| = | equal sign |
| $ | currency sign |
| , | comma (decimal point) |
| ; | semicolon |
| . | period (decimal point, full stop) |
| " | quotation mark |
| ( | left parenthesis |
| ) | right parenthesis |
| > | greater than symbol |
| < | less than symbol |
| : | colon |

*** COBOL word**. See “word.”

**code page**. An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for 8-bit code, assignment of characters and meanings to 128 code points for 7-bit code.

*** collating sequence**. The sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging, comparing, and for processing indexed files sequentially.

*** column**. A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

*** combined condition**. A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

*** comment-entry**. An entry in the IDENTIFICATION DIVISION that may be any combination of characters from the computer's character set.

*** comment line**. A source program line represented by an asterisk (*) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a slant (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection prior to printing the comment.

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

*** compile**. (1) To translate a program expressed in a high-level language into a program expressed in an intermediate language, assembly language, or a computer language. (2) To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler.

*** compile time**. The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

**compiler**. A program that translates a program written in a higher level language into a machine language object program.

**compiler directing statement**. A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

**compiler directing statement**. A statement that specifies actions to be taken by the compiler during processing of a COBOL source program. Compiler directives are contained in the COBOL source program. Thus, you can specify different suboptions of the directive within the source program by using multiple compiler directive statements in the program.

*** complex condition**. A condition in which one or more logical operators act upon one or more conditions. (See also "negated simple condition," "combined condition," and "negated combined condition.")

*** computer-name**. A system-name that identifies the computer upon which the program is to be compiled or run.

**condition**. An exception that has been enabled, or recognized, by Language Environment and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware/operating system and results in an interrupt. They can also be detected by language-specific generated code or language library code.

*** condition**. A status of a program at run time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2,...) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2,...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

*** conditional expression**. A simple condition or a complex condition specified in an EVALUATE, IF, PERFORM, or SEARCH statement. (See also "simple condition" and "complex condition.")

*** 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.

*** conditional statement**. A statement specifying that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

*** conditional variable**. A data item one or more values of which has a condition-name assigned to it.

*** condition-name**. A user-defined word that assigns a name to a subset of values that a conditional variable may assume; or a user-defined word assigned to a status of an implementor defined switch or device. When 'condition-name' is used in the general formats, it represents a unique data item reference consisting of a syntactically correct combination of a 'condition-name', together with qualifiers and subscripts, as required for uniqueness of reference.

*** condition-name condition**. The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values

attributed to a condition-name associated with the conditional variable.

*** CONFIGURATION SECTION**. A section of the ENVIRONMENT DIVISION that describes overall specifications of source and object programs and class definitions.

**CONSOLE**. A COBOL environment-name associated with the operator console.

*** contiguous items**. Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

**copybook**. A file or library member containing a sequence of code that is included in the source program at compile time using the COPY statement. The file can be created by the user, supplied by COBOL, or supplied by another product.

**CORBA**. The Common Object Request Broker Architecture established by the Object Management Group. IBM's *Interface Definition Language* used to describe the *interface* for SOM classes is fully compliant with CORBA standards.

*** counter**. A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

**cross-reference listing**. The portion of the compiler listing that contains information on where files, fields, and indicators are defined, referenced, and modified in a program.

**currency sign**. The character '$' of the COBOL character set or that character defined by the CURRENCY compiler option. If the NOCURRENCY compiler option is in effect, the currency sign is defined as the character '$'.

**currency symbol**. The character defined by the CURRENCY compiler option or by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If the NOCURRENCY compiler option is in effect for a COBOL source program and the CURRENCY SIGN clause is also **not** present in the source program, the currency symbol is identical to the currency sign.

*** current record**. In file processing, the record that is available in the record area associated with a file.

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

# D

*** data clause**. A clause, appearing in a data description entry in the DATA DIVISION of a COBOL program, that provides information describing a particular attribute of a data item.

*** data description entry** . An entry in the DATA DIVISION of a COBOL program that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

**DATA DIVISION**. One of the four main components of a COBOL program, class definition, or method definition. The DATA DIVISION describes the data to be processed by the object program, class, or method: files to be used and the records contained within them; internal working-storage records that will be needed; data to be made available in more than one program in the COBOL run unit. (Note, the Class DATA DIVISION contains only the WORKING-STORAGE SECTION.)

*** data item**. A unit of data (excluding literals) defined by a COBOL program or by the rules for function evaluation.

*** data-name**. A user-defined word that names a data item described in a data description entry. When used in the general formats, 'data-name' represents a word that must not be reference-modified, subscripted or qualified unless specifically permitted by the rules for the format.

**DBCS (Double-Byte Character Set)**. See "Double-Byte Character Set (DBCS)."

*** debugging line**. A debugging line is any line with a 'D' in the indicator area of the line.

*** debugging section**. A section that contains a USE FOR DEBUGGING statement.

*** declarative sentence**. A compiler directing sentence consisting of a single USE statement terminated by the separator period.

*** declaratives**. A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is com-

posed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one, or more associated paragraphs.

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

*** delimited scope statement**. Any statement that includes its explicit scope terminator.

*** delimiter**. A character or a sequence of contiguous characters that identify the end of a string of characters and separate that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

*** descending key**. A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

**digit**. Any of the numerals from 0 through 9. In COBOL, the term is not used in reference to any other symbol.

*** digit position**. The amount of physical storage required to store a single digit. This amount may vary depending on the usage specified in the data description entry that defines the data item.

*** direct access**. The facility to obtain data from storage devices or to enter data into a storage device in such a way that the process depends only on the location of that data and not on a reference to data previously accessed.

*** division**. A collection of zero, one or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four (4) divisions in a COBOL program: Identification, Environment, Data, and Procedure.

*** division header**. A combination of words followed by a separator period that indicates the beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

**do construction**. In structured programming, a DO statement is used to group a number of statements in a procedure. In COBOL, an in-line PERFORM statement functions in the same way.

**do-until**. In structured programming, a do-until loop will be executed at least once, and until a given condition is true. In COBOL, a TEST AFTER phrase used with the PERFORM statement functions in the same way.

**do-while**. In structured programming, a do-while loop will be executed if, and while, a given condition is true. In COBOL, a TEST BEFORE phrase used with the PERFORM statement functions in the same way.

**Double-Byte Character Set (DBCS)**. A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require Double-Byte Character Sets. Because each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS-capable.

*** dynamic access**. An access mode in which specific logical records can be obtained from or placed into a mass storage file in a nonsequential manner and obtained from a file in a sequential manner during the scope of the same OPEN statement.

**Dynamic Storage Area (DSA)**. Dynamically acquired storage composed of a register save area and an area available for dynamic storage allocation (such as program variables). DSAs are generally allocated within STACK segments managed by Language Environment.

# E

*** EBCDIC (Extended Binary-Coded Decimal Interchange Code)**. A coded character set consisting of 8-bit coded characters.

**EBCDIC character**. Any one of the symbols included in the 8-bit EBCDIC (Extended Binary-Coded-Decimal Interchange Code) set.

**edited data item**. A data item that has been modified by suppressing zeroes and/or inserting editing characters.

*** editing character**. A single character or a fixed two-character combination belonging to the following set:

| Character | Meaning |
|---|---|
| b | space |
| 0 | zero |

| | |
|---|---|
| + | plus |
| – | minus |
| CR | credit |
| DB | debit |
| Z | zero suppress |
| * | check protect |
| $ | currency sign |
| , | comma (decimal point) |
| . | period (decimal point) |
| / | slant (virgule, slash) |

**element (text element)**. One logical unit of a string of text, such as the description of a single data item or verb, preceded by a unique code identifying the element type.

*** elementary item**. A data item that is described as not being further logically subdivided.

**enclave**. When running under the Language Environment product, an enclave is analogous to a run unit. An enclave can create other enclaves on MVS and CMS by a LINK, on CMS by CMSCALL, and the use of the system () function of C.

***end class header**. A combination of words, followed by a separator period, that indicates the end of a COBOL class definition. The end class header is:

```
END CLASS class-name.
```

***end method header**. A combination of words, followed by a separator period, that indicates the end of a COBOL method definition. The end method header is:

```
END METHOD method-name.
```

*** end of Procedure Division**. The physical position of a COBOL source program after which no further procedures appear.

*** end program header**. A combination of words, followed by a separator period, that indicates the end of a COBOL source program. The end program header is:

```
END PROGRAM program-name.
```

*** entry**. Any descriptive set of consecutive clauses terminated by a separator period and written in the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, or DATA DIVISION of a COBOL program.

*** environment clause**. A clause that appears as part of an ENVIRONMENT DIVISION entry.

**ENVIRONMENT DIVISION**. One of the four main component parts of a COBOL program, class definition, or method definition. The ENVIRONMENT DIVISION describes the computers upon which the source program is compiled and those on which the object program is executed, and provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

**environment-name**. A name, specified by IBM, that identifies system logical units, printer and card punch control characters, report codes, and/or program switches. When an environment-name is associated with a mnemonic-name in the ENVIRONMENT DIVISION, the mnemonic-name may then be substituted in any format in which such substitution is valid.

**environment variable**. Any of a number of variables that describe the way an operating system is going to run and the devices it is going to recognize.

**execution time**. See “run time.”

**execution-time environment**. See “run-time environment.”

*** explicit scope terminator**. A reserved word that terminates the scope of a particular Procedure Division statement.

**exponent**. A number, indicating the power to which another number (the base) is to be raised. Positive exponents denote multiplication, negative exponents denote division, fractional exponents denote a root of a quantity. In COBOL, an exponential expression is indicated with the symbol ‘**’ followed by the exponent.

*** expression**. An arithmetic or conditional expression.

*** extend mode**. The state of a file after execution of an OPEN statement, with the EXTEND phrase specified for that file, and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

**extensions**. Certain COBOL syntax and semantics supported by IBM compilers in addition to those described in ANSI Standard.

*** external data**. The data described in a program as external data items and external file connectors.

*** external data item**. A data item which is described as part of an external record in one or more programs of a run unit and which itself may be referenced from any program in which it is described.

*** external data record**. A logical record which is described in one or more programs of a run unit and

whose constituent data items may be referenced from any program in which they are described.

**external decimal item**. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte. Bits 0 through 3 of all other bytes contain 1's (hex F). For example, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. (Also know as "zoned decimal item.")

*** external file connector**. A file connector which is accessible to one or more object programs in the run unit.

**external floating-point item**. A format for representing numbers in which a real number is represented by a pair of distinct numerals. In a floating-point representation, the real number is the product of the fixed-point part (the first numeral), and a value obtained by raising the implicit floating-point base to a power denoted by the exponent (the second numeral).

For example, a floating-point representation of the number 0.0001234 is: 0.1234 -3, where 0.1234 is the mantissa and -3 is the exponent.

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

# F

*** figurative constant**. A compiler-generated value referenced through the use of certain reserved words.

*** file**. A collection of logical records.

*** file attribute conflict condition**. An unsuccessful attempt has been made to execute an input-output operation on a file and the file attributes, as specified for that file in the program, do not match the fixed attributes for that file.

*** file clause**. A clause that appears as part of any of the following DATA DIVISION entries: file description entry (FD entry) and sort-merge file description entry (SD entry).

*** file connector**. A storage area which 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.

**File-Control**. The name of an ENVIRONMENT DIVISION paragraph in which the data files for a given source program are declared.

*** file control entry**. A SELECT clause and all its subordinate clauses which declare the relevant physical attributes of a file.

*** file description entry**. An entry in the File Section of the DATA DIVISION that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

*** file-name**. A user-defined word that names a file connector described in a file description entry or a sort-merge file description entry within the File Section of the DATA DIVISION.

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

***file position indicator**. A conceptual entity that contains the value of the current key within the key of reference for an indexed file, or the record number of the current record for a sequential file, or the relative record number of the current record for a relative file, or indicates that no next logical record exists, or that an optional input file is not present, or that the at end condition already exists, or that no valid next record has been established.

*** File Section**. The section of the DATA DIVISION that contains file description entries and sort-merge file description entries together with their associated record descriptions.

**file system**. The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

*** fixed file attributes**. Information about a file which is established when a file is created and cannot subsequently be changed during the existence of the file. These attributes include the organization of the file (sequential, relative, or indexed), the prime record key, the alternate record keys, the code set, the minimum and maximum record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the blocking factor, the padding character, and the record delimiter.

*** fixed length record**. A record associated with a file whose file description or sort-merge description entry requires that all records contain the same number of character positions.

**fixed-point number**. A numeric data item defined with a PICTURE clause that specifies the location of an optional sign, the number of digits it contains, and the location of an optional decimal point. The format may be either binary, packed decimal, or external decimal.

**floating-point number**. A numeric data item containing a fraction and an exponent. Its value is obtained by multiplying the fraction by the base of the numeric data item raised to the power specified by the exponent.

*** format**. A specific arrangement of a set of data.

*** function**. A temporary data item whose value is determined at the time the function is referenced during the execution of a statement.

*** function-identifier**. A syntactically correct combination of character-strings and separators that references a function. The data item represented by a function is uniquely identified by a function-name with its arguments, if any. A function-identifier may include a reference-modifier. A function-identifier that references an alphanumeric function may be specified anywhere in the general formats that an identifier may be specified, subject to certain restrictions. A function-identifier that references an integer or numeric function may be referenced anywhere in the general formats that an arithmetic expression may be specified.

**function-name**. A word that names the mechanism whose invocation, along with required arguments, determines the value of a function.

# G

*** global name**. A name which is declared in only one program but which may be referenced from that program and from any program contained within that program. Condition-names, data-names, file-names, record-names, report-names, and some special registers may be global names.

*** group item**. A data item that is composed of subordinate data items.

# H

**header label**. (1) A file label or data set label that precedes the data records on a unit of recording media. (2) Synonym for beginning-of-file label.

*** high order end**. The leftmost character of a string of characters.

# I

**IBM COBOL extension**. Certain COBOL syntax and semantics supported by IBM compilers in addition to those described in ANSI Standard.

**IDENTIFICATION DIVISION**. One of the four main component parts of a COBOL program, class definition, or method definition. The IDENTIFICATION DIVISION identifies the program name, class name, or method name. The IDENTIFICATION DIVISION may include the following documentation: author name, installation, or date.

*** identifier**. A syntactically correct combination of character-strings and separators that names a data item. When referencing a data item that is not a function, an identifier consists of a data-name, together with its qualifiers, subscripts, and reference-modifier, as required for uniqueness of reference. When referencing a data item which is a function, a function-identifier is used.

**IGZCBSN**. The COBOL for MVS & VM bootstrap routine. It must be link-edited with any module that contains a COBOL for MVS & VM program.

*** imperative statement**. A statement that either begins with an imperative verb and specifies an unconditional action to be taken or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements.

*** implicit scope terminator**. A separator period which terminates the scope of any preceding unterminated statement, or a phrase of a statement which by its occurrence indicates the end of the scope of any statement contained within the preceding phrase.

*** index**. A computer storage area or register, the content of which represents the identification of a particular element in a table.

*** index data item**. A data item in which the values associated with an index-name can be stored in a form specified by the implementor.

**indexed data-name**. An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

*** indexed file**. A file with indexed organization.

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

**indexing**. Synonymous with subscripting using index-names.

*** index-name**. A user-defined word that names an index associated with a specific table.

*** inheritance (for classes)**. A mechanism for using the implementation of one or more *classes* as the basis for another class. A *sub-class* inherits from one or more *super-classes*. By definition the inheriting class conforms to the inherited classes.

*** initial program**. A program that is placed into an initial state every time the program is called in a run unit.

*** initial state**. The state of a program when it is first called in a run unit.

**inline**. In a program, instructions that are executed sequentially, without branching to routines, subroutines, or other programs.

*** input file**. A file that is opened in the INPUT mode.

*** input mode**. The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

*** input-output file**. A file that is opened in the I-O mode.

*** INPUT-OUTPUT SECTION**. The section of the ENVIRONMENT DIVISION that names the files and the external media required by an object program or method and that provides information required for transmission and handling of data during execution of the object program or method definition.

*** Input-Output statement**. A statement that causes files to be processed by performing operations upon individual records or upon the file as a unit. The input-output statements are: ACCEPT (with the identifier phrase), CLOSE, DELETE, DISPLAY, OPEN, READ, REWRITE, SET (with the TO ON or TO OFF phrase), START, and WRITE.

*** input procedure**. A set of statements, to which control is given during the execution of a SORT statement, for the purpose of controlling the release of specified records to be sorted.

**instance data**. Data defining the state of an object. The instance data introduced by a class is defined in the WORKING-STORAGE SECTION of the DATA DIVISION of the class definition. The state of an object also includes the state of the instance variables introduced by base classes that are inherited by the current class. A separate copy of the instance data is created for each object instance.

*** integer**. (1) A numeric literal that does not include any digit positions to the right of the decimal point.

(2) A numeric data item defined in the DATA DIVISION that does not include any digit positions to the right of the decimal point.

(3) A numeric function whose definition provides that all digits to the right of the decimal point are zero in the returned value for any possible evaluation of the function.

**integer function**. A function whose category is numeric and whose definition does not include any digit positions to the right of the decimal point.

**interface**. The information that a *client* must know to use a *class*—the names of its *attributes* and the signatures of its *methods*. With direct-to-SOM compilers such as COBOL, the interface to a class may be defined by native language syntax for class definitions. Classes implemented in other languages might have their interfaces defined directly in SOM Interface Definition Language (IDL). The COBOL compiler has a compiler option, IDLGEN, to automatically generate IDL for a COBOL class.

**Interface Definition Language (IDL)**. The formal language (independent of any programming language) by which the *interface* for a class of *objects* is defined in a IDL file, which the SOM compiler then interprets to create an implementation template file and binding files. SOM's Interface Definition Language is fully compliant with standards established by the Object Management Group's Common Object Request Broker Architecture (*CORBA*).

**interlanguage communication (ILC)**. The ability of routines written in different programming languages to communicate. ILC support allows the application writer to readily build applications from component routines written in a variety of languages.

**intermediate result**. An intermediate field containing the results of a succession of arithmetic operations.

*** internal data**. The data described in a program excluding all external data items and external file connectors. Items described in the LINKAGE SECTION of a program are treated as internal data.

*** internal data item**. A data item which is described in one program in a run unit. An internal data item may have a global name.

**internal decimal item**. A format in which each byte in a field except the rightmost byte represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111. (Also known as packed decimal.)

*** internal file connector**. A file connector which is accessible to only one object program in the run unit.

*** intra-record data structure**. The entire collection of groups and elementary data items from a logical record which is defined by a contiguous subset of the data description entries which describe that record. These data description entries include all entries whose level-number is greater than the level-number of the first data description entry describing the intra-record data structure.

**intrinsic function**. A pre-defined function, such as a commonly used arithmetic function, called by a built-in function reference.

*** invalid key condition**. A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

*** I-O-CONTROL**. The name of an ENVIRONMENT DIVISION paragraph in which object program requirements for rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

*** I-O-CONTROL entry**. An entry in the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION which contains clauses that provide information required for the transmission and handling of data on named files during the execution of a program.

*** I-O-Mode**. The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phase for that file.

*** I-O status**. A conceptual entity which contains the two-character value indicating the resulting status of an input-output operation. This value is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

**iteration structure**. A program processing logic in which a series of statements is repeated while a condition is true or until a condition is true.

# K

**K**. When referring to storage capacity, two to the tenth power; 1024 in decimal notation.

*** key**. A data item that identifies the location of a record, or a set of data items which serve to identify the ordering of data.

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

*** key word**. A reserved word or function-name whose presence is required when the format in which the word appears is used in a source program.

**kilobyte (KB)**. One kilobyte equals 1024 bytes.

# L

*** language-name**. A system-name that specifies a particular programming language.

**Language Environment-conforming**. A characteristic of compiler products (COBOL/370, COBOL for MVS & VM, AD/Cycle C/370, C/C++ for MVS and VM, PL/I for MVS and VM) that produce object code conforming to the Language Environment format.

**last-used state**. A program is in last-used state if its internal values remain the same as when the program was exited (are not reset to their initial values).

*** letter**. A character belonging to one of the following two sets:

1. Uppercase letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
2. Lowercase letters: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

*** level indicator**. Two alphabetic characters that identify a specific type of file or a position in a hierarchy. The level indicators in the DATA DIVISION are: CD, FD, and SD.

*** level-number**. A user-defined word, expressed as a two digit number, which indicates the hierarchical position of a data item or the special properties of a data description entry. Level-numbers in the range from 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77 and 88 identify special properties of a data description entry.

*** library-name**. A user-defined word that names a COBOL library that is to be used by the compiler for a given source program compilation.

*** library text**. A sequence of text words, comment lines, the separator space, or the separator pseudo-text delimiter in a COBOL library.

**LILIAN DATE**. The number of days since the beginning of the Gregorian calendar. Day one is Friday, October 15, 1582. The Lilian date format is named in honor of Luigi Lilio, the creator of the Gregorian calendar.

*** LINAGE-COUNTER**. A special register whose value points to the current position within the page body.

**LINKAGE SECTION**. The section in the DATA DIVISION of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

**literal**. A character-string whose value is specified either by the ordered set of characters comprising the string, or by the use of a figurative constant.

**local**. A set of attributes for a program execution environment indicating culturally sensitive considerations, such as: character code page, collating sequence, date/time format, monetary value representation, numeric value representation, or language.

*** LOCAL-STORAGE SECTION**. The section of the DATA DIVISION that defines storage that is allocated and freed on a per-invocation basis, depending on the value assigned in their VALUE clauses.

*** logical operator**. One of the reserved words AND, OR, or NOT. In the formation of a condition, either AND, or OR, or both can be used as logical connectives. NOT can be used for logical negation.

*** logical record**. The most inclusive data item. The level-number for a record is 01. A record may be either an elementary item or a group of items. The term is synonymous with record.

*** low order end**. The rightmost character of a string of characters.

# M

**main program**. In a hierarchy of programs and subroutines, the first program to receive control when the programs are run.

*** mass storage**. A storage medium in which data may be organized and maintained in both a sequential and nonsequential manner.

*** mass storage device**. A device having a large storage capacity; for example, magnetic disk, magnetic drum.

*** mass storage file**. A collection of records that is assigned to a mass storage medium.

*** megabyte (M)**. One megabyte equals 1,048,576 bytes.

*** merge file**. A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

**metaclass**. A SOM class whose instances are SOM class-objects. The methods defined in metaclasses are executed without requiring any object instances of the class to exist, and are frequently used to create instances of the class.

**method**. Procedural code that defines one of the operations supported by an object, and that is executed by an INVOKE statement on that object.

*** Method Definition**. The COBOL source unit that defines a method.

*** method identification entry**. An entry in the METHOD-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the method-name and assign selected attributes to the method definition.

*** method-name**. A user-defined word that identifies a method.

*** mnemonic-name**. A user-defined word that is associated in the ENVIRONMENT DIVISION with a specified implementor-name.

**multitasking**. Mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks. When running under the Language Environment product, multitasking is synonymous with *multithreading*.

**MVS/XA* (Multiple Virtual Storage/Extended Architecture)**. An IBM operating system that manages multiple virtual address spaces in IBM processors operating in extended architecture mode. MVS/XA supports the 31-bit addressing mechanism of extended architecture mode, and therefore, allows an address space as large as $2^{31}$ bytes (2048 megabytes or 2 gigabytes).

# N

**name**. A word composed of not more than 30 characters that defines a COBOL operand.

*** native character set**. The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

*** native collating sequence**. The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

*** negated combined condition**. The 'NOT' logical operator immediately followed by a parenthesized combined condition.

*** negated simple condition**. The 'NOT' logical operator immediately followed by a simple condition.

**nested program**. A program that is directly contained within another program.

*** next executable sentence**. The next sentence to which control will be transferred after execution of the current statement is complete.

*** next executable statement**. The next statement to which control will be transferred after execution of the current statement is complete.

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

*** noncontiguous items**. Elementary data items in the WORKING-STORAGE and LINKAGE SECTIONs that bear no hierarchic relationship to other data items.

*** nonnumeric item**. A data item whose description permits its content to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

*** nonnumeric literal**. A literal bounded by quotation marks. The string of characters may include any character in the computer's character set.

**null**. Figurative constant used to assign the value of an invalid address to pointer data items. NULLS can be used wherever NULL can be used.

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

**numeric-edited item**. A numeric item that is in such a form that it may be used in printed output. It may consist of external decimal digits from 0 through 9, the decimal point, commas, the dollar sign, editing sign control symbols, plus other editing symbols.

*** numeric function**. A function whose class and category are numeric but which for some possible evaluation does not satisfy the requirements of integer functions.

*** numeric item**. A data item whose description restricts its content to a value represented by characters chosen from the digits from '0' through '9'; if signed, the item may also contain a '+', '–', or other representation of an operational sign.

*** numeric literal**. A literal composed of one or more numeric characters that may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

# O

**object**. An entity that has state (its data values) and operations (its methods). An object is a way to encapsulate state and behavior.

**object code**. Output from a compiler or assembler that is itself executable machine code or is suitable for processing to produce executable machine code.

*** OBJECT-COMPUTER**. The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the object program is executed, is described.

*** object computer entry**. An entry in the OBJECT-COMPUTER paragraph of the ENVIRONMENT DIVISION which contains clauses that describe the computer environment in which the object program is to be executed.

**object deck**. A portion of an object program suitable as input to a linkage editor. Synonymous with *object module* and *text deck*.

**object module**. Synonym for *object deck* or *text deck*.

*** object of entry**. A set of operands and reserved words, within a DATA DIVISION entry of a COBOL program, that immediately follows the subject of the entry.

*** object program**. A set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'object program.'

*** object time**. The time at which an object program is executed. The term is synonymous with execution time.

*** obsolete element**. A COBOL language element in Standard COBOL that is to be deleted from the next revision of Standard COBOL.

**ODBC**. Open Database Connectivity that provides you access to data from a variety of databases and file systems.

**ODO object**. In the example below,

```
WORKING-STORAGE SECTION
01  TABLE-1.
    05  X                   PICS9.
    05  Y OCCURS 3 TIMES
          DEPENDING ON X    PIC X.
```

`X` is the object of the OCCURS DEPENDING ON clause (ODO object). The value of the ODO object determines how many of the ODO subject appear in the table.

**ODO subject**. In the example above, `Y` is the subject of the OCCURS DEPENDING ON clause (ODO subject). The number of `Y` ODO subjects that appear in the table depends on the value of `X`.

*** open mode**. The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O or EXTEND.

*** operand**. Whereas the general definition of operand is "that component which is operated upon," for the purposes of this document, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

*** operational sign**. An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

*** optional file**. A file which is declared as being not necessarily present each time the object program is executed. The object program causes an interrogation for the presence or absence of the file.

*** optional word**. A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

**OS/2 (Operating System/2*)**. A multi-tasking operating system for the IBM Personal Computer family that allows you to run both DOS mode and OS/2 mode programs.

*** output file**. A file that is opened in either the OUTPUT mode or EXTEND mode.

*** output mode**. The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

*** output procedure**. A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

**overflow condition**. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

# P

**packed decimal item**. See "internal decimal item."

*** padding character**. An alphanumeric character used to fill the unused character positions in a physical record.

**page**. A vertical division of output data representing a physical separation of such data, the separation being based on internal logical requirements and/or external characteristics of the output medium.

*** page body**. That part of the logical page in which lines can be written and/or spaced.

*** paragraph**. In the Procedure Division, a paragraph-name followed by a separator period and by zero, one, or more sentences. In the IDENTIFICATION and ENVIRONMENT DIVISIONs, a paragraph header followed by zero, one, or more entries.

*** paragraph header**. A reserved word, followed by the separator period, that indicates the beginning of a paragraph in the IDENTIFICATION and ENVIRONMENT DIVISIONs. The permissible paragraph headers in the IDENTIFICATION DIVISION are:

```
PROGRAM-ID. (Program IDENTIFICATION DIVISION)
CLASS-ID. (Class IDENTIFICATION DIVISION)
METHOD-ID. (Method IDENTIFICATION DIVISION)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

The permissible paragraph headers in the ENVIRONMENT DIVISION are:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
REPOSITORY. (Program or Class CONFIGURATION SECTION)
FILE-CONTROL.
I-O-CONTROL.
```

*** paragraph-name**. A user-defined word that identifies and begins a paragraph in the Procedure Division.

**parameter**. Parameters are used to pass data values between calling and called programs.

**password**. A unique string of characters that a program, computer operator, or user must supply to meet security requirements before gaining access to data.

*** phrase**. A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

*** physical record**. See "block."

**pointer data item**. A data item in which address values can be stored. Data items are explicitly defined as pointers with the USAGE IS POINTER clause. ADDRESS OF special registers are implicitly defined as pointer data items. Pointer data items can be compared for equality or moved to other pointer data items.

**portability**. The ability to transfer an application program from one application platform to another with relatively few changes to the source program.

**preloaded**. In COBOL this refers to COBOL programs that remain resident in storage under IMS instead of being loaded each time they are called.

*** prime record key**. A key whose contents uniquely identify a record within an indexed file.

*** priority-number**. A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters '0','1', ... , '9'. A segment-number may be expressed either as a one- or two-digit number.

*** procedure**. A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

*** 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 in the source program. The procedure branching statements are: ALTER, CALL, EXIT, EXIT PROGRAM, GO TO, MERGE, (with the OUTPUT PROCEDURE phrase), PERFORM and SORT (with the INPUT PROCEDURE or OUTPUT PROCEDURE phrase).

**Procedure Division**. One of the four main component parts of a COBOL program, class definition, or method definition. The Procedure Division contains instructions for solving a problem. The Program and Method Procedure Divisions may contain imperative statements, conditional statements, compiler directing statements, paragraphs, procedures, and sections. The Class Procedure Division contains only method definitions.

**procedure integration**. One of the functions of the COBOL optimizer is to simplify calls to performed procedures or contained programs.

PERFORM procedure integration is the process whereby a PERFORM statement is replaced by its performed procedures. Contained program procedure integration is the process where a CALL to a contained program is replaced by the program code.

* **procedure-name**. A user-defined word that is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which may be qualified) or a section-name.

**procedure-pointer data item**. A data item in which a pointer to an entry point can be stored. A data item defined with the USAGE IS PROCEDURE-POINTER clause contains the address of a procedure entry point.

* **program identification entry**. An entry in the PROGRAM-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the program-name and assign selected program attributes to the program.

* **program-name**. In the IDENTIFICATION DIVISION and the end program header, a user-defined word that identifies a COBOL source program.

* **pseudo-text**. A sequence of text words, comment lines, or the separator space in a source program or COBOL library bounded by, but not including, pseudo-text delimiters.

* **pseudo-text delimiter**. Two contiguous equal sign characters (==) used to delimit pseudo-text.

* **punctuation character**. A character that belongs to the following set:

| Character | Meaning |
|---|---|
| , | comma |
| ; | semicolon |
| : | colon |
| . | period (full stop) |
| " | quotation mark |
| ( | left parenthesis |
| ) | right parenthesis |
| b | space |
| = | equal sign |

# Q

**QSAM (Queued Sequential Access Method)**. An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or of output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

* **qualified data-name**. An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

* **qualifier**.

1. A data-name or a name associated with a level indicator which is used in a reference either together with another data-name which is the name of an item that is subordinate to the qualifier or together with a condition-name.
2. A section-name that is used in a reference together with a paragraph-name specified in that section.
3. A library-name that is used in a reference together with a text-name associated with that library.

# R

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

* **record**. See “logical record.”

* **record area**. A storage area allocated for the purpose of processing the record described in a record description entry in the File Section of the DATA DIVISION. In the File Section, the current number of character positions in the record area is determined by the explicit or implicit RECORD clause.

* **record description**. See “record description entry.”

* **record description entry**. The total set of data description entries associated with a particular record. The term is synonymous with record description.

**recording mode**. The format of the logical records in a file. Recording mode can be F (fixed-length), V (variable-length), S (spanned), or U (undefined).

**record key**. A key whose contents identify a record within an indexed file.

*** record-name**. A user-defined word that names a record described in a record description entry in the DATA DIVISION of a COBOL program.

*** record number**. The ordinal number of a record in the file whose organization is sequential.

**recursion**. A program calling itself or being directly or indirectly called by a one of its called programs.

**recursively capable**. A program is recursively capable (can be called recursively) if the RECURSIVE attribute is on the PROGRAM-ID statement.

**reel**. A discrete portion of a storage medium, the dimensions of which are determined by each implementor that contains part of a file, all of a file, or any number of files. The term is synonymous with unit and volume.

**reentrant**. The attribute of a program or routine that allows more than one user to share a single copy of a load module.

*** reference format**. A format that provides a standard method for describing COBOL source programs.

**reference modification**. A method of defining a new alphanumeric data item by specifying the leftmost character and length relative to the leftmost character of another alphanumeric data item.

*** reference-modifier**. A syntactically correct combination of character-strings and separators that defines a unique data item. It includes a delimiting left parenthesis separator, the leftmost character position, a colon separator, optionally a length, and a delimiting right parenthesis separator.

*** relation**. See “relational operator” or “relation condition.”

*** relational operator**. A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

| Operator | Meaning |
|---|---|
| IS GREATER THAN | Greater than |
| IS > | Greater than |
| IS NOT GREATER THAN | Not greater than |
| IS NOT > | Not greater than |
| | |
| IS LESS THAN | Less than |
| IS < | Less than |
| IS NOT LESS THAN | Not less than |
| IS NOT < | Not less than |
| | |
| IS EQUAL TO | Equal to |
| IS = | Equal to |
| IS NOT EQUAL TO | Not equal to |
| IS NOT = | Not equal to |
| | |
| IS GREATER THAN OR EQUAL TO | Greater than or equal to |
| IS >= | Greater than or equal to |
| | |
| IS LESS THAN OR EQUAL TO | Less than or equal to |
| IS <= | Less than or equal to |

*** relation character**. A character that belongs to the following set:

| Character | Meaning |
|---|---|
| | |
| > | greater than |
| < | less than |
| = | equal to |

*** relation condition**. The proposition, for which a truth value can be determined, that the value of an arithmetic expression, data item, nonnumeric literal, or index-name has a specific relationship to the value of another arithmetic expression, data item, nonnumeric literal, or index name. (See also “relational operator.”)

*** relative file**. A file with relative organization.

*** relative key**. A key whose contents identify a logical record in a relative file.

*** relative organization**. The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record’s logical ordinal position in the file.

*** relative record number**. The ordinal number of a record in a file whose organization is relative. This number is treated as a numeric literal which is an integer.

* **reserved word**. A COBOL word specified in the list of words that may be used in a COBOL source program, but that must not appear in the program as user-defined words or system-names.

* **resource**. A facility or service, controlled by the operating system, that can be used by an executing program.

* **resultant identifier**. A user-defined data item that is to contain the result of an arithmetic operation.

**reusable environment**. A reusable environment is when you establish an assembler program as the main program by using either ILBOSTP0 programs, IGZERRE programs, or the RTEREUS run-time option.

**routine**. A set of statements in a COBOL program that causes the computer to perform an operation or series of related operations. In Language Environment, refers to either a procedure, function, or subroutine.

* **routine-name**. A user-defined word that identifies a procedure written in a language other than COBOL.

* **run time**. The time at which an object program is executed. The term is synonymous with object time.

**run-time environment**. The environment in which a COBOL program executes.

* **run unit**. A stand-alone object program, or several object programs, that interact via COBOL CALL statements, which function at run time as an entity.

# S

**SBCS (Single Byte Character Set)**. See "Single Byte Character Set (SBCS)".

**scope terminator**. A COBOL reserved word that marks the end of certain Procedure Division statements. It may be either explicit (END-ADD, for example) or implicit (separator period).

* **section**. A set of zero, one or more paragraphs or entities, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

* **section header**. A combination of words followed by a separator period that indicates the beginning of a section in the Environment, Data, and Procedure Divisions. In the ENVIRONMENT and DATA DIVISIONs, a section header is composed of reserved words followed by a separator period. The permissible section headers in the ENVIRONMENT DIVISION are:

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

The permissible section headers in the DATA DIVISION are:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a separator period.

* **section-name**. A user-defined word that names a section in the Procedure Division.

**selection structure**. A program processing logic in which one or another series of statements is executed, depending on whether a condition is true or false.

* **sentence**. A sequence of one or more statements, the last of which is terminated by a separator period.

* **separately compiled program**. A program which, together with its contained programs, is compiled separately from all other programs.

* **separator**. A character or two contiguous characters used to delimit character-strings.

* **separator comma**. A comma (,) followed by a space used to delimit character-strings.

* **separator period**. A period (.) followed by a space used to delimit character-strings.

* **separator semicolon**. A semicolon (;) followed by a space used to delimit character-strings.

**sequence structure**. A program processing logic in which a series of statements is executed in sequential order.

* **sequential access**. An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

* **sequential file**. A file with sequential organization.

* **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.

**serial search**. A search in which the members of a set are consecutively examined, beginning with the first member and ending with the last.

*** 77-level-description-entry**. A data description entry that describes a noncontiguous data item with the level-number 77.

*** sign condition**. The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

*** simple condition**. Any single condition chosen from the set:

Relation condition
Class condition
Condition-name condition
Switch-status condition
Sign condition

**Single Byte Character Set (SBCS)**. A set of characters in which each character is represented by a single byte. See also "EBCDIC (Extended Binary-Coded Decimal Interchange Code)."

**slack bytes**. Bytes inserted between data items or records to ensure correct alignment of some numeric items. Slack bytes contain no meaningful data. In some cases, they are inserted by the compiler; in others, it is the responsibility of the programmer to insert them. The SYNCHRONIZED clause instructs the compiler to insert slack bytes when they are needed for proper alignment. Slack bytes between records are inserted by the programmer.

**SOM**. *System Object Model*

*** sort file**. A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

*** sort-merge file description entry**. An entry in the File Section of the DATA DIVISION that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

*** SOURCE-COMPUTER**. The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the source program is compiled, is described.

*** source computer entry**. An entry in the SOURCE-COMPUTER paragraph of the ENVIRONMENT DIVISION which contains clauses that describe the computer environment in which the source program is to be compiled.

*** source item**. An identifier designated by a SOURCE clause that provides the value of a printable item.

**source program**. Although it is recognized that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements. A COBOL source program commences with the IDENTIFICATION DIVISION or a COPY statement. A COBOL source program is terminated by the end program header, if specified, or by the absence of additional source program lines.

*** special character**. A character that belongs to the following set:

| Character | Meaning |
|---|---|
| + | plus sign |
| – | minus sign (hyphen) |
| * | asterisk |
| / | slant (virgule, slash) |
| = | equal sign |
| $ | currency sign |
| , | comma (decimal point) |
| ; | semicolon |
| . | period (decimal point, full stop) |
| " | quotation mark |
| ( | left parenthesis |
| ) | right parenthesis |
| > | greater than symbol |
| < | less than symbol |
| : | colon |

*** special-character word**. A reserved word that is an arithmetic operator or a relation character.

**SPECIAL-NAMES**. The name of an ENVIRONMENT DIVISION paragraph in which environment-names are related to user-specified mnemonic-names.

*** special names entry**. An entry in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION which provides means for specifying the currency sign; choosing the decimal point; specifying symbolic characters; relating implementor-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters.

*** special registers**. Certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of a specific COBOL feature.

*** standard data format**. The concept used in describing the characteristics of data in a COBOL DATA DIVISION under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

*** statement**. A syntactically valid combination of words, literals, and separators, beginning with a verb, written in a COBOL source program.

**STL File System**. STandard Language File System: native workstation and PC file system for COBOL and PL/I. Supports sequential, relative, and indexed files, including the full ANSI 85 COBOL standard I/O language and all of the extensions described in *IBM COBOL Language Reference*, unless exceptions are explicitly noted.

**structured programming**. A technique for organizing and coding a computer program in which the program comprises a hierarchy of segments, each segment having a single entry point and a single exit point. Control is passed downward through the structure without unconditional branches to higher levels of the hierarchy.

*** sub-class**. A class that inherits from another class. When two classes in an inheritance relationship are considered together, the sub-class is the inheritor or inheriting class; the *super-class* is the inheritee or inherited class.

*** subject of entry**. An operand or reserved word that appears immediately following the level indicator or the level-number in a DATA DIVISION entry.

*** subprogram**. See “called program.”

*** subscript**. An occurrence number represented by either an integer, a data-name optionally followed by an integer with the operator + or -, or an index-name optionally followed by an integer with the operator + or -, that identifies a particular element in a table. A subscript may be the word ALL when the subscripted identifier is used as a function argument for a function allowing a variable number of arguments.

*** subscripted data-name**. An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

*** super-class**. A class that is inherited by another class. See also *sub-class*.

**switch-status condition**. The proposition, for which a truth value can be determined, that an UPSI switch, capable of being set to an ‘on’ or ‘off’ status, has been set to a specific status.

*** symbolic-character**. A user-defined word that specifies a user-defined figurative constant.

**syntax**. (1) The relationship among characters or groups of characters, independent of their meanings or the manner of their interpretation and use. (2) The structure of expressions in a language. (3) The rules governing the structure of a language. (4) The relationship among symbols. (5) The rules for the construction of a statement.

*** system-name**. A COBOL word that is used to communicate with the operating environment.

**System Object Model (SOM)**. IBM's object-oriented programming technology for building, packaging, and manipulating class libraries. SOM conforms to the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) standards.

# T

*** table**. A set of logically consecutive items of data that are defined in the DATA DIVISION by means of the OCCURS clause.

*** table element**. A data item that belongs to the set of repeated items comprising a table.

**text deck**. Synonym for *object deck* or *object module*.

*** text-name**. A user-defined word that identifies library text.

*** text word**. A character or a sequence of contiguous characters between margin A and margin R in a COBOL library, source program, or in pseudo-text which is:

- A separator, except for: space; a pseudo-text delimiter; and the opening and closing delimiters for non-numeric literals. The right parenthesis and left parenthesis characters, regardless of context within

the library, source program, or pseudo-text, are always considered text words.

- A literal including, in the case of nonnumeric literals, the opening quotation mark and the closing quotation mark that bound the literal.
- Any other sequence of contiguous COBOL characters except comment lines and the word 'COPY' bounded by separators that are neither a separator nor a literal.

**top-down design**. The design of a computer program using a hierarchic structure in which related functions are performed at each level of the structure.

**top-down development**. See "structured programming."

**trailer-label**. (1) A file or data set label that follows the data records on a unit of recording medium. (2) Synonym for end-of-file label.

*** truth value**. The representation of the result of the evaluation of a condition in terms of one of two values: true or false.

# U

*** unary operator**. A plus (+) or a minus (-) sign, that precedes a variable or a left parenthesis in an arithmetic expression and that has the effect of multiplying the expression by +1 or -1, respectively.

**unit**. A module of direct access, the dimensions of which are determined by IBM.

**universal object reference**. A data-name that can refer to an object of any class.

*** unsuccessful execution**. The attempted execution of a statement that does not result in the execution of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referenced by that statement, but may affect status indicators.

**UPSI switch**. A program switch that performs the functions of a hardware switch. Eight are provided: UPSI-0 through UPSI-7.

*** user-defined word**. A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

# V

*** variable**. A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

*** variable length record**. A record associated with a file whose file description or sort-merge description entry permits records to contain a varying number of character positions.

*** variable occurrence data item**. A variable occurrence data item is a table element which is repeated a variable number of times. Such an item must contain an OCCURS DEPENDING ON clause in its data description entry, or be subordinate to such an item.

*** variably located group.**. A group item following, and not subordinate to, a variable-length table in the same level-01 record.

*** variably located item.**. A data item following, and not subordinate to, a variable-length table in the same level-01 record.

*** verb**. A word that expresses an action to be taken by a COBOL compiler or object program.

**VM/SP (Virtual Machine/System Product)**. An IBM-licensed program that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a "real" machine.

**volume**. A module of external storage. For tape devices it is a reel; for direct-access devices it is a unit.

**volume switch procedures**. System specific procedures executed automatically when the end of a unit or reel has been reached before end-of-file has been reached.

# W

*** word**. A character-string of not more than 30 characters which forms a user-defined word, a system-name, a reserved word, or a function-name.

*** WORKING-STORAGE SECTION**. The section of the DATA DIVISION that describes working storage data items, composed either of noncontiguous items or working storage records or of both.

# Z

**zoned decimal item**. See “external decimal item.”

# Index

## Special Characters


, (comma)
  insertion character 181
  symbol in PICTURE clause 172, 176
/ (slash)
  insertion character 181
  symbol in PICTURE clause 172, 176
(/) comment line 40
. (period) 173
{ : }
  description 31
  required use of 492
$ (currency)
  insertion character 182, 183
  symbol in PICTURE clause 173, 176
* symbol in PICTURE clause 173
*CBL (*CONTROL) statement 484
+ (plus)
  insertion character 182, 183, 184
  SIGN clause 192
  symbol in PICTURE clause 176
– (minus)
  insertion character 182, 183
  SIGN clause 192
  symbol in PICTURE clause 176
< (less than) 230
<= (less than or equal to) 230
= (equal) 230
> (greater than) 230
>= (greater than or equal to) 230


## Numerics


0
  insertion character 181
  symbol in PICTURE clause 172, 176
66, RENAMES data description entry 190
77, item description entry 136
88, condition-name data description entry 158
9, symbol in PICTURE clause 172, 176


## A


A, symbol in PICTURE clause 171
abbreviated combined relation condition
  examples 244
  using parentheses in 242
abbreviated combined relation condition *(continued)*
ACCEPT statement
  mnemonic name in 261, 262
  overlapping operands, unpredictable results 253
  system information transfer 263
  under AIX and OS/2 261
access mode
  description 112
  dynamic
    DELETE statement 287
    description 113
    READ statement 365
  random
    DELETE statement 287
    description 113
    READ statement 364
  sequential
    DELETE statement 286
    description 113
    READ statement 362
ACCESS MODE clause 112
ACOS function 435
ADD statement
  common phrases 249
  CORRESPONDING phrase 267
  description and format 266
ADDRESS OF special register 13
ADDRESS OF under AIX and OS/2 384
ADVANCING phrase 417
AFTER phrase
  INSPECT statement 316
  PERFORM statement 349
  with REPLACING 313
  with TALLYING 312
  WRITE statement 417
AIX COBOL language differences
  ACCEPT statement 261
  APPLY WRITE-ONLY 126
  ASSIGN clause 104
  basis-names, library-names, text-names 42
  BLOCK CONTAINS 145
  CLOSE statement 279
  CODE-SET 156
  COMP-5 data item 202



© Copyright IBM Corp. 1991, 1996

AIX COBOL language differences *(continued)*
compiler limits 518
control characters 138
DBCS 4
environment-name 263
file handling 126
file status data-name-8 119
LABEL RECORDS 150
LINE SEQUENTIAL file I-O 110
locale definition 543
LOCK MODE clause 117
MORE-LABELS, GO TO 304
pointer data item size 205
RECORD CONTAINS clause 148
RECORDING MODE clause 154
RERUN clause 122
RESERVE clause 107
SAME AREA clause 123
SAME SORT AREA clause 125
SAME SORT-MERGE AREA clause 125
SET statement 384
SORT-CONTROL special register 18
SORT-FILE-SIZE under AIX and OS/2 20
SORT-MESSAGE under AIX and OS/2 20
SORT-MODE-SIZE under AIX and OS/2 21
status key value and meaning 255
USE...AFTER...LABEL PROCEDURE 510
WRITE ADVANCING 90, 417
aligning data 193
ALL
phrase of INSPECT statement 312, 313
SEARCH statement 378
UNSTRING statement 408
ALL literal
STOP statement 398
STRING statement 400
UNSTRING statement 408
ALL Subscripting 430
ALPHABET clause 91
alphabet-name
description 91
MERGE statement 329
PROGRAM COLLATING SEQUENCE clause 86
SORT statement 390
alphabetic character in ACCEPT 261
alphabetic class and category 136
ALPHABETIC class test 226
alphabetic item
alignment rules 137
elementary move rules 335
alphabetic item *(continued)*
PICTURE clause 176
ALPHABETIC-LOWER class test 227
ALPHABETIC-UPPER class test 227
alphanumeric arguments 428
alphanumeric class and category
alignment rules 137
description 136
alphanumeric edited item
alignment rules 137
elementary move rules 335
PICTURE clause 178
alphanumeric functions 427
alphanumeric item
alignment rules 137
elementary move rules 335
PICTURE clause 178
alphanumeric literal, control character restrictions 138
ALSO phrase
ALPHABET clause 93
EVALUATE statement 295
ALTER statement
description and format 268
GO TO statement and 304
segmentation considerations 269
altered GO TO statement 304
ALTERNATE RECORD KEY clause 115
AND logical operator 238
ANNUITY function 436
APPLY WRITE-ONLY clause 126
Area A (cols. 8-11) 35
Area B (cols. 12-72) 37
arguments 428
arithmetic expression
COMPUTE statement 283
description 223
EVALUATE statement 296
relation condition 229
arithmetic operator
description 224
permissible symbol pairs 225
arithmetic statements
ADD 266
common phrases 249
COMPUTE 283
DIVIDE 291
list of 252
multiple results 253
MULTIPLY 338
operands 252

arithmetic statements *(continued)*
programming notes 253
SUBTRACT 405
ASCENDING KEY phrase
collating sequence 165
description 328
MERGE statement 328
OCCURS clause 164
SORT statement 389
ASCII
collating sequence 526
processing considerations 539
specifying in SPECIAL-NAMES paragraph 91
ASIN function 437
ASSIGN clause
description 103
format 99
SELECT clause and 103
assigning index values 381
assignment-name
ASSIGN clause 103
RERUN clause 122
under AIX and OS/2 104
asterisk (*)
comment line 40
insertion character 184
at end condition
READ statement 364
RETURN statement 369
AT END phrase
READ statement 360
RETURN statement 369
SEARCH statement 375
AT END-OF-PAGE phrases 419
ATAN function 438
AUTHOR paragraph
description 81
format 72

# B

B
insertion character 181
symbol in PICTURE clause 171
BASIS statement 482
basis-names on AIX and OS/2 42
batch compile 64
BEFORE phrase
INSPECT statement 316
PERFORM statement 349
BEFORE phrase *(continued)*
with REPLACING 313
with TALLYING 312
WRITE statement 417
binary arithmetic operators 224
binary data item, DISPLAY statement 288
BINARY phrase in USAGE clause 200
binary search 378
blank line 40
BLANK WHEN ZERO clause
description and format 160
USAGE IS INDEX clause 203
BLOCK CONTAINS clause
description 145
format 140
branching
GO TO statement 303
out-of-line PERFORM statement 347
BY CONTENT phrase
CALL statement 272
on INVOKE statement 321
BY REFERENCE phrase
CALL statement 272
on INVOKE statement 320
BY VALUE phrase
CALL statement 273
on INVOKE statement 322

# C

C01-C012 under AIX and OS/2 417
call convention 514
CALL statement
CANCEL statement and 277
description and format 270
Linkage Section 220
ON OVERFLOW phrase 270
Procedure Division header 217, 219
program termination statements 270
subprogram linkage 270
transfer of control 59
USING phrase 219
called and calling programs, description 270
CALLINTERFACE directive 514
CANCEL statement 277
carriage control character 418
category of data
alphabetic items 176
alphanumeric items 178
alphanumeric-edited items 178

category of data *(continued)*
DBCS items 179
numeric items 177
numeric-edited items 177
relationship to class of data 136
CBL (PROCESS) statement 483
CHAR function 439
character code set, specifying 91
character-string
COBOL word 5
representation in PICTURE clause 176
size determination 138
CHARACTERS BY phrase 313
CHARACTERS phrase
BLOCK CONTAINS clause 146
INSPECT statement 312
MEMORY SIZE clause 86
USAGE clause and 146
characters, valid in COBOL program 4
checkpoint processing, RERUN clause 122
class 68
CLASS clause 94
class condition 226, 227
class Data Division 128
class definition
affect of SELF and SUPER 319
class procedure division 215
CLASS-ID paragraph 77
Configuration Section 84
Data Division 128
description 68
Identification Division 74
inheritance rules 78
requirements for indexed tables 166
class identification division 77
class name, OO 43
class procedure division 215
CLASS-ID paragraph 77
class-name class test 227
classes of data 136
clauses 32, 33
CLOSE statement
affect on record locking 118
format and description 279
COBOL
class definition 68
language structure 4
method definition 70
program structure 62
reference format 34
COBOL 74 Standard 548
COBOL 85 Standard
definition 548
COBOL word 5
CODE-SET clause
ALPHABET clause and 93
description 156
format 140
NATIVE phrase and 156
collating sequence
ASCENDING/DESCENDING KEY phrase and 165
ASCII 526
EBCDIC 522
locale definition 543
specified in OBJECT-COMPUTER paragraph 86
specified in SPECIAL-NAMES paragraph 91
COLLATING SEQUENCE phrase 86
ALPHABET clause 91
MERGE statement 329
SORT statement 390
colon character
description 31
required use of 492
column 7
indicator area 37
specifying comments 39
combined condition
description 240
evaluation rules 241
logical operators and evaluation results 240
order of evaluation 241
permissible element sequences 240
comma (,)
Configuration Section 84
DECIMAL-POINT IS COMMA clause 96
insertion character 181
symbol in PICTURE clause 172
comment line
description 39
Identification Division 81
in library text 488
COMMON clause 76
common processing facilities 254
COMP-1 through COMP-5 data items 201
comparison
cycle, INSPECT statement 317
DBCS operands 237
in EVALUATE statement 297
nonnumeric operands 234
numeric and nonnumeric operands 236

comparison *(continued)*
numeric operands 232
of index data items 236
of index-names 236
rules for COPY statement 489
compile-time switch 530
compiler directing statements
BASIS 482
COPY 486
DELETE 493
description 485
EJECT 495
ENTER 496
INSERT 497
READY TRACE 498
REPLACE 500
RESET TRACE 498
SERVICE LABEL 504
SERVICE RELOAD 505
SKIP1/2/3 506
TITLE 507
USE 508
compiler limits 518
compiler options
ADV 418
controlling output from 484
NUMPROC 238
specifying 483
THREAD 166
TRUNC 139
complex conditions
abbreviated combined relation 242
combined condition 240
description 238
negated simple 239
complex OCCURS DEPENDING ON (CODO) 169
composite of operands 252
COMPUTATIONAL data items 201
COMPUTE statement
common phrases 250
description and format 283
computer-name 85, 86
condition
abbreviated combined relation 242
class 226
combined 240
complex 238
condition-name 228
EVALUATE statement 296
IF statement 305
condition *(continued)*
negated simple 239
PERFORM UNTIL statement 349
relation 229
SEARCH statement 376
sign 237
simple 225
switch-status 238
condition-name
and conditional variable 158
description and format 228
rules for values 210
SEARCH statement 379
SET statement 383
SPECIAL-NAMES paragraph 91
switch status condition 91
conditional expression
comparing index-names and index data items 236
comparison of DBCS operands 237
description 225
order of evaluation of operands 241
parentheses in abbreviated combined relation conditions 242
conditional statements
description 246
GO TO statement 303
IF statement 305
list of 246
PERFORM statement 349
conditional variable 158
Configuration Section
description (programs, classes, methods) 84
REPOSITORY paragraph 96
SOURCE-COMPUTER paragraph 85
SPECIAL-NAMES paragraph 88
conformance rules
general rules 80
INVOKE...USING 323
multiple inheritance 78
SET...USAGE OBJECT REFERENCE 387
Contained Programs 62
continuation
area 34
lines 37, 39
CONTINUE statement 285
control characters, using on AIX or OS/2 138
CONTROL statement (*CONTROL) 484
control transfer 58
conversion of data, DISPLAY statement 288

CONVERTING phrase 315
COPY statement
comparison rules 489
description and format 486
example 491
replacement rules 489
REPLACING phrase 488
SUPPRESS option 488
CORRESPONDING (CORR) phrase
ADD statement 267
description 267
MOVE statement 333
SUBTRACT statement 406
with ON SIZE ERROR phrase 251
COS function 440
COUNT IN phrase, UNSTRING statement 409
CR (credit)
insertion character 182
symbol in PICTURE clause 173
CURRENCY SIGN clause 95
currency symbol 173
currency symbol ($) 182
CURRENT-DATE function 441

# D

data
alignment 137
categories 137, 176
classes 136
format of standard 138
hierarchies used in qualification 133
organization 108
signed 139
truncation of 138, 162
data category
alphabetic items 176
alphanumeric items 178
alphanumeric-edited items 178
DBCS items 179
numeric items 177
numeric-edited items 177
data conversion, DISPLAY statement 288
data description entry
BLANK WHEN ZERO clause 160
data-name 159
description and format 157
FILLER phrase 159
GLOBAL clause 161
indentation and 136
data description entry *(continued)*
JUSTIFIED clause 162
level-66 format (previously defined items) 158
level-88 format (condition-names) 158
level-number description 158
OCCURS clause 163
OCCURS DEPENDING ON (ODO) clause 166
PICTURE clause 169
REDEFINES clause 186
RENAMES clause 190
SIGN clause 192
SYNCHRONIZED clause 193
USAGE clause 200
VALUE clause 207
DATA DIVISION
ASCII considerations 541
data description entry 157
data relationships 133
data types 132
description (programs, classes, methods) 128
file description (FD) entry 144
levels of data 133
Linkage Section 131
sort description (SD) entry 144
Working-Storage Section 129
data flow
STRING statement 402
UNSTRING statement 412
data item
data description entry 157
description entry definition 129
EXTERNAL clause 160
record description entry 157
data manipulation statements
ACCEPT 261
INITIALIZE 307
list of 253
MOVE 333
overlapping operands 254
READ 358
RELEASE 366
RETURN 368
REWRITE 370
SET 381
STRING 400
UNSTRING 407
WRITE 415
data organization
access modes and 113
indexed 109

data organization *(continued)*
line sequential 109
relative 109
sequential 108
DATA RECORDS clause
description 151
format 140
data transfer 261
data types
file data 132
program data 133
data-name
data description entry 159
data-names
precedence if duplicate 128
DATE 264
DATE-COMPILED paragraph
description 81
format 72
DATE-OF-INTEGER function 443
DATE-WRITTEN paragraph
description 81
format 72
DAY 264
DAY-OF-INTEGER function 444
DAY-OF-WEEK 265
DB (debit)
insertion character 182
symbol in PICTURE clause 173
DBCS (Double-Byte Character Set)
class and category 136
elementary move rules 336
PICTURE clause and 179
use with relational operators 230
using in comments 82
DBCS class condition 227
De-editing 335
DEBUG-ITEM special register 13
debugging 529
DEBUGGING declarative 512
debugging line 40, 85
DEBUGGING MODE clause 85
decimal point (.) 250
DECIMAL-POINT IS COMMA clause
description 96
declarative procedures
description and format 220
PERFORM statement 346
USE statement 220
declaratives
EXCEPTION/ERROR 508
LABEL 510
precedence rules for nested programs 510
USE FOR DEBUGGING 512
DECLARATIVES key word
begin in Area A 36
description 220
Declaratives Section 220
DELETE statement
affect on record locking 118
description and format 493
dynamic access 287
format and description 286
INVALID KEY phrases 287
random access 287
sequential access 286
DELIMITED BY phrase
STRING 401
UNSTRING statement 407
delimited scope statement 247
delimiter
INSPECT statement 314
UNSTRING statement 407
DELIMITER IN phrase, UNSTRING statement 409
DEPENDING phrase
GO TO statement 303
OCCURS clause 166
DESCENDING KEY phrase 164
collating sequence 165
description 328
MERGE statement 328
SORT statement 389
DISPLAY phrase in USAGE clause 202
DISPLAY statement
description and format 288
external 138, 179
programming notes 290
DIVIDE statement
common phrases 250
description and format 291
REMAINDER phrase 293
division header
format, Environment Division 84
format, Identification Division 72
format, Procedure Division 217
specification of 35
DO-UNTIL structure, PERFORM statement 349
DO-WHILE structure, PERFORM statement 349

Double-Byte Character Set (DBCS)
  class and category 136
  PICTURE clause and 179
  use with relational operators 230
  using in comments 82
DOWN BY phrase, SET statement 382
duplicate data-names, precedence 128
DUPLICATES phrase
  KEY phrase 395
  SORT statement 390
  START statement 395
dynamic access mode
  data organization and 113
  DELETE statement 287
  description 113
  READ statement 365

# E

E, symbol in PICTURE clause 171
EBCDIC
  CODE-SET clause and 156
  collating sequence 522
  specifying in SPECIAL-NAMES paragraph 91
editing
  fixed insertion 182
  floating insertion 183
  replacement 184
  signs 139
  simple insertion 181
  special insertion 181
  suppression 184
editing sign control symbol 173
eject page 40
EJECT statement 495
elementary item
  alignment rules 137
  basic subdivisions of a record 133
  classes and categories 136
  MOVE statement 334
  nonnumeric operand comparison 236
  size determination in program 138
  size determination in storage 138
elementary move rules 334
ELSE NEXT SENTENCE phrase 305
END DECLARATIVES key word 220
END PROGRAM 64
end program header 36
END-CALL phrase 276
END-IF phrase 305
END-INVOKE phrase 325
end-of-file processing 279
END-OF-PAGE phrases 419
END-PERFORM phrase 348
ENTER statement 496
entry
  definition 32
ENTRY statement
  description and format 294
  subprogram linkage 294
Environment Division
  ASCII considerations 539
  compiler limits 518
  Configuration Section
    ALPHABET clause 91
    CURRENCY SIGN clause 95
    OBJECT-COMPUTER paragraph 86
    REPOSITORY paragraph 96
    SOURCE-COMPUTER paragraph 85
    SPECIAL-NAMES paragraph 88, 94
    SYMBOLIC CHARACTERS clause 93
  Input-Output Section
    FILE-CONTROL paragraph 99
  REPOSITORY paragraph 96
environment names with WRITE ADVANCING 417
environment-name
  SPECIAL-NAMES paragraph 90
environment-name under AIX and OS/2 263
EOP phrases 419
equal sign (=) 229
EQUAL TO relational operator 229
ERROR declarative statement 508
EUC
  description 4
EVALUATE statement
  comparing operands 297
  determining truth value 296
  format and description 295
evaluation rules
  combined conditions 241
  EVALUATE statement 297
  nested IF statement 306
EXCEPTION declarative statement 508
EXCEPTION/ERROR declarative
  CLOSE statement 280
  DELETE statement 286
  description and format 508
execution flow
  ALTER statement changes 268

execution flow *(continued)*
PERFORM statement changes 346
EXIT METHOD statement
format and description 300
EXIT PROGRAM statement
format and description 301
EXIT statement
format and description 299
PERFORM statement 347
explicit
scope terminators 248
exponentiation
exponential expression 223
expression, arithmetic 223
EXTEND phrase
OPEN statement 341
EXTERNAL clause
with data item 160
with file name 144
external decimal item
DISPLAY statement 288
external floating point
alignment rules 138
DISPLAY statement 288
PICTURE clause and 179

# F

FACTORIAL function 445
FALSE phrase 296
FD (File Description) entry
BLOCK CONTAINS clause 145
DATA RECORDS clause 151
description 144
format 140
LABEL RECORDS clause 150
level indicator 133
VALUE OF clause 151
figurative constant
DISPLAY statement 289
STOP statement 398
STRING statement 400
symbolic-character 11
UNSTRING statement 408
file
data type 132
definition 132
handling, on AIX and OS/2 126
labels 150
file description entry
GLOBAL clause 145
file organization
definition 113
LINAGE clause 151
line sequential 109
types of 108
file position indicator
description 260
READ statement 363
File Section
EXTERNAL clause 144
for object-oriented programs 128
RECORD clause 147
file sharing 117
FILE STATUS clause
DELETE statement and 286
description 119
format 99
INVALID KEY phrase and 258
status key 254
FILE-CONTROL paragraph
ASSIGN clause 103
description and format 99
FILE STATUS clause 119
ORGANIZATION clause 108
PADDING CHARACTER clause 111
RECORD KEY clause 114
RELATIVE KEY clause 116
RESERVE clause 107
SELECT clause 103
file-name, specifying on SELECT clause 103
FILLER phrase
CORRESPONDING phrase 159
data description entry 159
fixed insertion editing 182
fixed-length
item, maximum length 157
records 145
floating insertion editing 183
floating-point
DISPLAY statement 288
internal 138
FOOTING phrase of LINAGE clause 151
FOR REMOVAL phrase 279, 280
format notation, rules for xiii
FROM phrase
ACCEPT statement 262
REWRITE statement 370
SUBTRACT statement 405

FROM phrase *(continued)*
with identifier 259
WRITE statement 416
function
arguments 428
class and category 137
description 426
rules for usage 428
types of functions 427

# G

G, symbol in PICTURE clause 171
GIVING phrase
ADD statement 266
arithmetic 250
DIVIDE statement 293
MERGE statement 330
MULTIPLY statement 338
SORT statement 392
SUBTRACT statement 406
GLOBAL clause 129, 161
GO TO statement
altered 304
conditional 303
format and description 303
MORE-LABELS 304
SEARCH statement 375
unconditional 303
GOBACK statement 302
GREATER THAN OR EQUAL TO symbol (>=) 229
GREATER THAN symbol (>) 229
group item
class and categories 136
description 133
MOVE statement 337
nonnumeric operand comparison 236
group move rules 337

# H

halting execution 398
HIGH-VALUE(S) figurative constant 93
hyphen (-), in indicator area 37

# I

IBM extensions, format description xiii
Identification Division
CLASS-ID paragraph 77
Identification Division *(continued)*
format (program, class, method) 72
METHOD-ID paragraph 79
optional paragraphs 81
PROGRAM-ID paragraph 75
identifier 48, 222, 223
IF statement 305
imperative statement 245
implicit
redefinition of storage area 144, 186
scope terminators 248
in line PERFORM statement 346
indentation 37, 136
index
data item 236, 333
relative indexing 54
SET statement 54
index name
assigning values 381
comparisons 236
data item definition 203
OCCURS clause 166
PERFORM statement 356
SEARCH statement 374
SET statement 381
INDEX phrase in USAGE clause 203
INDEXED BY phrase 166
indexed files
CLOSE statement 280
DELETE statement 287
FILE-CONTROL paragraph format 99
I-O-CONTROL paragraph format 121
organization 109
permissible statements for 345
READ statement 363
START statement 396
indexed organization
description 109
FILE-CONTROL paragraph format 99
I-O-CONTROL paragraph format 121
indexing
description 53
MOVE statement evaluation 333
OCCURS clause 53, 163
relative 54
SET statement and 54
indicator area 34
industry specifications 546
INHERITS clause 77

INITIAL clause 76
initial state of program 76
INITIALIZE statement
  format and description 307
  overlapping operands, unpredictable results 253
input file, label processing 343
Input-Output Section
  description 98
  FILE-CONTROL paragraph 99
  format 98
  I-O-CONTROL paragraph 120
input-output statements
  ACCEPT 261
  CLOSE 279
  common processing facilities 254
  DELETE 286
  DISPLAY 288
  EXCEPTION/ERROR procedures 509
  general description 254
  OPEN 340
  READ 358
  REWRITE 370
  START 395
  WRITE 415
INPUT phrase
  OPEN statement 341
  USE statement 508
INPUT PROCEDURE phrase
  RELEASE statement 366
  SORT statement 392
Input-Output Section
  on AIX and OS/2 110
insertion editing
  fixed (numeric-edited items) 182
  floating (numeric-edited items) 183
  simple 181
  special (numeric-edited items) 181
INSPECT statement
  AFTER phrase 314
  BEFORE phrase 314
  comparison cycle 317
  CONVERTING phrase 315
  overlapping operands, unpredictable results 253
  REPLACING phrase 312
INSTALLATION paragraph
  description 81
  format 72
instance data 70
integer arguments 428
INTEGER function 446
Integer functions 427
INTEGER-OF-DATE function 447
INTEGER-OF-DAY function 448
INTEGER-PART function 449
internal floating-point
  alignment rules 138
  DISPLAY statement 288
INTO phrase
  DIVIDE statement 291
  READ statement 358
  RETURN statement 368
  STRING statement 401
  UNSTRING statement 409
  with identifier 259
intrinsic functions
  ACOS 435
  alphanumeric function 427
  ANNUITY 436
  ASIN 437
  ATAN 438
  CHAR 439
  COS 440
  CURRENT-DATE 441
  DATE-OF-INTEGER 443
  DAY-OF-INTEGER 444
  FACTORIAL 445
  floating-point literals 430
  INTEGER 446
  integer function 427
  INTEGER-OF-DATE 447
  INTEGER-OF-DAY 448
  INTEGER-PART 449
  LENGTH 450
  LOG 451
  LOG10 452
  LOWER-CASE 453
  MAX 454
  MEAN 455
  MEDIAN 456
  MIDRANGE 457
  MIN 458
  MOD 459
  numeric function 427
  NUMVAL 460
  NUMVAL-C 461
  ORD 463
  ORD-MAX 464
  ORD-MIN 465
  PRESENT-VALUE 466

intrinsic functions *(continued)*
RANDOM 467
RANGE 468
REM 469
REVERSE 470
SIN 471
SQRT 472
STANDARD-DEVIATION 473
SUM 474
summary of 432
TAN 475
UPPER-CASE 476
VARIANCE 477
WHEN-COMPILED 478
invalid key condition 258
INVALID KEY phrase
DELETE statement 287
READ statement 360
REWRITE statement 371
START statement 396
WRITE statement 420
INVOKE statement
BY CONTENT phrase 321
BY REFERENCE phrase 320
BY VALUE phrase 322
format and description 319
NOT ON EXCEPTION phrase 325
ON EXCEPTION phrase 325
RETURNING phrase 324
USING phrase 320
I-O-CONTROL paragraph
APPLY WRITE-ONLY clause 126
checkpoint processing in 122
description 98, 120
MULTIPLE FILE TAPE clause 126
order of entries 121
RERUN clause 122
SAME AREA clause 123
SAME RECORD AREA clause 124
SAME SORT AREA clause 125
SAME SORT-MERGE AREA clause 125
ISCII processing considerations 539

# J

JUSTIFIED clause
description and format 162
effect on initial settings 162
STRING statement 401
truncation of data 162
JUSTIFIED clause *(continued)*
USAGE IS INDEX clause and 162
VALUE clause and 208

# K

Kanji 227
key of reference 109
KEY phrase
OCCURS clause 164
READ statement 360
SEARCH statement 374
SORT statement 389
START statement 395

# L

LABEL declarative 510
label processing, OPEN statement 343
LABEL RECORDS clause
description 150
format 140
Language Environment Callable Services
description 270
LEADING phrase
INSPECT statement 312, 313
SIGN clause 192
LENGTH function 450
LENGTH OF special register 15
LESS THAN OR EQUAL TO symbol (<=) 229
LESS THAN symbol (<) 229
level
01 item 134
02-49 item 134
66 item 136
77 item 136
88 item 136
indicator, definition of 133
level number
definition 133
description and format 158
FILLER phrase 159
library-name
COPY statement 486
library-names
on AIX and OS/2 42
limits of the compiler 518
LINAGE clause
description 151
diagram of phrases 152

LINAGE clause *(continued)*
format 140
LINAGE-COUNTER special register
description 16
WRITE statement 418
line advancing 417
line sequential file organization 109
LINE/LINES, WRITE statement 417
LINES AT BOTTOM phrase 152
LINES AT TOP phrase 152
Linkage Section
called subprogram 220
description 131
levels under AIX and OS/2 384
requirement for indexed items 166
VALUE clause 207
literal
and arithmetic expressions 223
ASSIGN clause 103
CODE-SET clause and ALPHABET clause 93
CURRENCY SIGN clause 95
description 22
nonnumeric operand comparison 236
null-terminated nonnumeric 25
STOP statement 398
VALUE clause 208
local storage
defining with RECURSIVE clause 75
requirement for indexed items 166
Local-Storage
object-oriented COBOL 128
locale 543
LOCK MODE clause 117
LOG function 451
LOG10 function 452
logical operator
complex condition 238
in evaluation of combined conditions 240
list of 238
logical record
definition 132
file data 132
program data 133
record description entry and 132
RECORDS phrase 146
LOW-VALUE(S) figurative constant 93
LOWER-CASE function 453

# M

MAX function 454
maximum index value 54
MEAN function 455
MEDIAN function 456
MEMORY SIZE clause 86
MERGE statement
ASCENDING/DESCENDING KEY phrase 328
COLLATING SEQUENCE phrase 329
format and description 327
GIVING phrase 330
OUTPUT PROCEDURE phrase 331
USING phrase 330
METACLASS clause 77, 203
metaclass, description 68
method data division 128
method definition
affect of SELF and SUPER 319
Data Division 128
description 70
Identification Division 74
inheritance rules 78
method procedure division 215
METHOD-ID paragraph 79
method identification division 79
method name 42
method procedure division 215
METHOD-ID paragraph 79
methods
available to subclasses 78
exiting 300
invoking 319
recursively reentering 75
reusing 77
MIDRANGE function 457
MIN function 458
minus sign (-)
COBOL character 4
fixed insertion symbol 182
floating insertion symbol 183, 184
SIGN clause 192
mnemonic-name
ACCEPT statement 261
DISPLAY statement 289
SET statement 383
SPECIAL-NAMES paragraph 90
WRITE statement 417
MOD function 459
MORE-LABELS GO TO statement 304

MOVE statement
  CORRESPONDING phrase 333
  elementary moves 334
  format and description 333
  group moves 337
MULTIPLE FILE TAPE clause 126
multiple inheritance 78
multiple record processing, READ statement 361
multiple results, arithmetic statements 253
multiple volume files, treatment on AIX and OS/2 126
MULTIPLY statement
  common phrases 250
  format and description 338
multivolume files
  READ statement 363
  WRITE statement 422

# N

native binary data item 202
native character set 91
native collating sequence 91
negated combined condition 240
negated simple condition 239
NEGATIVE 238
nested IF structure
  description 306
  EVALUATE statement 295
nested programs
  description 62
  precedence rules for 510
NEXT RECORD phrase, READ statement 358
NEXT SENTENCE phrase
  IF statement 305
  SEARCH statement 375
NO ADVANCING phrase, DISPLAY statement 289
NO REWIND phrase
  OPEN statement 341
  under AIX and OS/2 279
non-reel file, definition 280
nonnumeric literals 25
nonnumeric operands, comparing 234
NOT AT END phrase
  READ statement 360
  RETURN statement 369
NOT INVALID KEY phrase
  DELETE statement 287
  READ statement 360
  REWRITE statement 371
  START statement 396
NOT ON EXCEPTION phrase
  CALL statement 276
  on INVOKE statement 325
NOT ON OVERFLOW phrase
  STRING statement 402
  UNSTRING statement 411
NOT ON SIZE ERROR phrase
  ADD statement 267
  DIVIDE statement 293
  general description 251
  MULTIPLY statement 339
  SUBTRACT statement 406
NULL 211
null block branch, CONTINUE statement 285
null-terminated nonnumeric literals 25
numeric arguments 428
numeric class and category 136
NUMERIC class test 226
numeric edited item
  alignment rules 137
  editing signs 139
  elementary move rules 335
  PICTURE clause 177
numeric function 427
numeric item 177
numeric operands, comparing 232
NUMVAL function 460
NUMVAL-C function 461

# O

Object Program 62
OBJECT REFERENCE phrase 203
object time switch 530
OBJECT-COMPUTER paragraph 86
object-oriented COBOL
  class definition 68
  comparison rules 232
  conformance rules
    general rules 80
    INVOKE...USING 323
    multiple inheritance 78
    SET...USAGE OBJECT REFERENCE 387
  Data Division (class and method) 128
  effect of GLOBAL attribute 129
  Identification Division (class and method) 72
  INHERITS clause 77
  INVOKE statement 319
  method definition 70
  method name 42

object-oriented COBOL *(continued)*
multiple inheritance 78
OO class name 43
Procedure Division (class and method) 215
REPOSITORY paragraph 96
SELF and SUPER special characters 9
specifying configuration section 84
subclasses and methods 78
USAGE OBJECT REFERENCE clause 203
objects in EVALUATE statement 295
obsolete language elements xii
OCCURS clause
ASCENDING/DESCENDING KEY phrase 164
description 163
INDEXED BY phrase 166
restrictions 163
variable-length tables format 166
OCCURS DEPENDING ON (ODO) clause
complex 169
description 167
format 166
RECORD clause 147
REDEFINES clause and 163
SEARCH statement and 163
subject and object of 167
subject of 163
subscripting 51
OFF phrase, SET statement 383
ON EXCEPTION phrase
CALL statement 275
on INVOKE statement 325
ON OVERFLOW phrase
CALL statement 276
DISPLAY statement 290
STRING statement 401, 411
ON phrase, SET statement 383
ON SIZE ERROR phrase
ADD statement 267
arithmetic statements 251
COMPUTE statement 283
DIVIDE statement 293
MULTIPLY statement 339
SUBTRACT statement 406
OPEN statement
affect on record locking 118
format and description 340
I-O phrase 341
label processing 343
phrases 340
programming notes 343
OPEN statement *(continued)*
system dependencies 345
operands
comparison of nonnumeric 234
comparison of numeric 232
composite of 252
overlapping 253, 254
operational sign
algebraic, description of 139
SIGN clause and 139
USAGE clause and 139
ORD function 463
ORD-MAX function 464
ORD-MIN function 465
order of entries
clauses in FILE-CONTROL paragraph 99
IO CONTROL paragraph 121
order of evaluation in combined conditions 241
ORGANIZATION clause
description 108
format 99
ORGANIZATION IS INDEXED clause 108
ORGANIZATION IS LINE SEQUENTIAL clause 108
ORGANIZATION IS RELATIVE clause 108
ORGANIZATION IS SEQUENTIAL clause 108
OS/2 COBOL language differences
ACCEPT statement 261
APPLY WRITE-ONLY 126
ASSIGN clause 104
basis-names, library-names, text-names 42
BLOCK CONTAINS 145
CLOSE statement 279
CODE-SET clause 156
COMP-5 data item 202
compiler limits 518
control characters 138
DBCS 4
environment-name 263
file handling 126
file status data-name-8 119
LABEL RECORDS 150
LINE SEQUENTIAL file I-O 110
locale definition 543
LOCK MODE clause 117
MORE-LABELS, GO TO 304
pointer data item size 205
RECORD CONTAINS clause 148
RECORDING MODE clause 154
RERUN clause 122
RESERVE clause 107

OS/2 COBOL language differences *(continued)*
SAME AREA clause 123
SAME SORT AREA clause 125
SAME SORT-MERGE AREA clause 125
SET statement 384
SORT-CONTROL special register 18
SORT-FILE-SIZE under AIX and OS/2 20
SORT-MESSAGE under AIX and OS/2 20
SORT-MODE-SIZE under AIX and OS/2 21
specifying call convention 514
status key value and meaning 255
USAGE clause 203
USE...AFTER...LABEL PROCEDURE 510
WRITE ADVANCING 90, 417
out-of-line PERFORM statement 347
outermost programs, debugging 512
output file, label processing 343
OUTPUT phrase 341
OUTPUT PROCEDURE phrase
MERGE statement 331
RETURN statement 368
SORT statement 393
OVERFLOW phrase
CALL statement 276
STRING statement 401, 411
overlapping operands invalid in
arithmetic statements 253
data manipulation statements 254

# P

P, symbol in PICTURE clause 171
PACKED-DECIMAL phrase in USAGE clause 201
PADDING CHARACTER clause 111
page eject 40
paragraph
description 32, 222
header, specification of 35
termination, EXIT statement 299
paragraph name
description 222
specification of 35
parentheses
combined conditions, use 240
in arithmetic expressions 224
partial listings 484
PASSWORD clause
description 116
system dependencies 117
PERFORM statement
branching 347
conditional 349
END-PERFORM phrase 348
EVALUATE statement 295
execution sequences 348
EXIT statement 299
format and description 346
in-line 347
out-of-line 347
TIMES phrase 348
VARYING phrase 350, 353
period (.)
actual decimal point 181
phrase, definition 33
physical record
BLOCK CONTAINS clause 145
definition 132
file data 132
file description entry and 132
RECORDS phrase 146
PICTURE clause
and class condition 226
computational items and 201
CURRENCY SIGN clause 95
data categories in 176
DECIMAL-POINT IS COMMA clause 96, 170
description 169
editing 180
format 169
sequence of symbols 174
symbols used in 170
plus (+)
fixed insertion symbol 182
floating insertion symbol 183, 184
insertion character 184
SIGN clause 192
pointer data item
defined with USAGE clause 205
relation condition 231
SET statement 384
size on Workstation 205
POINTER phrase
STRING statement 401
UNSTRING statement 410
POSITIVE 238
PRESENT-VALUE function 466
PREVIOUS RECORD phrase, READ statement 358
print files, WRITE statement 422

procedure branching
GO TO statement 303
statements, executed sequentially 260
Procedure Branching Statements 260
Procedure Division
declarative procedures 220
format (programs, methods, classes) 215
header 217
statements 261
procedure-name
GO TO statement 303
MERGE statement 331
PERFORM statement 346
SORT statement 392
PROCEDURE-POINTER data item
defined with USAGE clause 206
relation condition 231
SET statement 385
size on Workstation 205
procedure, description 221
PROGRAM COLLATING SEQUENCE clause
ALPHABET clause 91
SPECIAL-NAMES paragraph and 86
program termination
GOBACK statement 302
STOP statement 398
PROGRAM-ID paragraph
description 75
format 72
program-name, rules for referencing 66
program, separately-compiled 62
programming notes
ACCEPT statement 261
altered GO TO statement 268
arithmetic statements 253
data manipulation statements 400, 407
DELETE statement 286
DISPLAY statement 290
EXCEPTION/ERROR procedures 510
OPEN statement 343
PERFORM statement 348
RECORDS clause 147
STRING statement 400
UNSTRING statement 407
programming structures 349
programs, recursive 75
pseudo-text
COPY statement operand 488
description 40
punch files, WRITE statement 422

# Q

quotation mark (") character 37

# R

railroad track format, how to read xiii
random access mode
data organization and 113
DELETE statement 287
description 113
READ statement 364
RANDOM function 467
RANGE function 468
READ statement
affect on record locking 118
AT END phrases 360
dynamic access mode 365
format and description 358
INTO identifier phrase 259, 358
INVALID KEY phrases 258, 360
KEY phrase 360
multiple record processing 361
multivolume files 363
NEXT RECORD phrase 358
overlapping operands, unpredictable results 253
programming notes 365
random access mode 364
READY TRACE statement 498
receiving field
COMPUTE statement 283
MOVE statement 333
multiple results rules 253
SET statement 381
STRING statement 401
UNSTRING statement 409
record
area description 147
elementary items 133
fixed-length 145
logical, definition of 132
physical, definition of 132
RECORD clause
description and format 147
omission of 147
RECORD CONTAINS 0 CHARACTERS 148
record description entry
levels of data 134

record description entry *(continued)*
logical record 132
RECORD KEY clause
description 114
format 99
record key in indexed file 287
record locking 117
RECORDING MODE clause 154
RECORDS phrase
BLOCK CONTAINS clause 146
RERUN clause 123
RECURSIVE clause 75
recursive methods 319
recursive programs 75
requirement for indexed items 166
REDEFINES clause
description 186
examples of 189
format 186
general considerations 188
OCCURS clause restriction 187
SYNCHRONIZED clause and 193
undefined results 189
VALUE clause and 187
redefinition, implicit 144
REEL phrase 279, 280
reference-modification 54, 57
reference-modifier
ALL Subscripting 430
Reference, methods of
Simple data 48
relation character
COPY statement 488
INITIALIZE statement 307
INSPECT statement 312
relation condition
abbreviated combined 242
comparison of numeric and nonnumeric operands 232
comparison with nonnumeric second operand 234
comparison with numeric second operand 233
description 229
operands of equal size 234
operands of unequal size 234
relational operator
in abbreviated combined relation condition 242
meaning of each 230
relation condition use 229
relative files
access modes allowed 114
relative files *(continued)*
CLOSE statement 280
DELETE statement 287
FILE-CONTROL paragraph format 99
I-O-CONTROL paragraph format 121
organization 109
permissible statements for 345
READ statement 362
RELATIVE KEY clause 114, 116
REWRITE statement 372
START statement 397
RELATIVE KEY clause
description 116
format 99
relative organization
access modes allowed 114
description 109
FILE-CONTROL paragraph format 99
I-O-CONTROL paragraph format 121
RELEASE statement 253, 366
REM function 469
REMAINDER phrase of DIVIDE statement 293
RENAMES clause
description and format 190
INITIALIZE statement 307
level 66 item 136, 190
PICTURE clause 169
REPLACE statement
comparison operation 501
continuation rules for pseudo-text 501
description and format 500
special notes 502
replacement editing 184
replacement rules for COPY statement 489
REPLACING phrase
COPY statement 488
INITIALIZE statement 308
REPOSITORY paragraph 96
required words xiii
RERUN clause
checkpoint processing 122
description 122
format 121
RECORDS phrase 122
sort/merge 123
under AIX and OS/2 122
RESERVE clause
description 107
format 99
under AIX and OS/2 107

reserved word list 532
RESET TRACE statement 498
result field
  GIVING phrase 250
  NOT ON SIZE ERROR phrase 251
  ON SIZE ERROR phrase 251
  ROUNDED phrase 250
RETURN CODE special register 17
RETURN statement
  AT END phrase 369
  description and format 368
  overlapping operands, unpredictable results 253
RETURNING phrase
  CALL statement 275
  on INVOKE statement 324
reusing logical records 371
REVERSE function 470
REWRITE statement
  affect on record locking 118
  description and format 370
  FROM identifier phrase 259
  INVALID KEY phrase 371
ROUNDED phrase
  ADD statement 267
  COMPUTE statement 283
  description 250
  DIVIDE statement 293
  MULTIPLY statement 339
  size error checking and 251
  SUBTRACT statement 406
rules for syntax notation xiii
Rules for Usage 428
run unit
  description 62
  termination with CANCEL statement 278

# S

S 172
S01-S05 environment names under AIX and OS/2 417
SAME AREA clause under AIX and OS/2 123
SAME clause 123
SAME RECORD AREA clause
  description 124
  format 121
SAME SORT AREA clause
  description 125
  format 121
SAME SORT AREA clause under AIX and OS/2 125
SAME SORT-MERGE AREA clause
  description 125
  format 121
scope terminator
  explicit 248
  implicit 248
SEARCH statement
  AT END phrase 375
  binary search 378
  description and format 374
  serial search 375
  SET statement 375
  USAGE IS INDEX clause 203
  VARYING phrase 376
  WHEN phrase 375
section 32, 221
section header
  description 221
  specification of 35
section name
  description 221
  in EXCEPTION/ERROR declarative 508
SECURITY paragraph
  description 81
  format 72
segmentation considerations 269
SELECT clause
  ASSIGN clause and 103
  format 99
  specifying a file name 103
SELECT OPTIONAL clause
  CLOSE statement 280
  description 103
  format 99
  specification for sequential I-O files 103
selection objects in EVALUATE statement 295
selection subjects in EVALUATE statement 295
SELF special character word 9, 319
sending field
  MOVE statement 333
  SET statement 381
  STRING statement 400
  UNSTRING statement 407
sentence
  COBOL, definition 33
  description 222
SEPARATE CHARACTER phrase of SIGN clause 192
separate sign, class condition 226
separately-compiled program 62

separator 210
sequence number area (cols. 1-6) 34
sequential access mode
  data organization and 113
  DELETE statement 286
  description 113
  READ statement 362
  REWRITE statement 371
sequential files
  access mode allowed 113
  CLOSE statement 279, 280
  description 108
  file description entry 140
  FILE-CONTROL paragraph format 99
  LINAGE clause 152
  OPEN statement 340
  PASSWORD clause valid with 116
  permissible statements for 344
  READ statement 362
  REWRITE statement 371
  SELECT OPTIONAL clause 103
serial search
  PERFORM statement 350
  SEARCH statement 375
SERVICE LABEL statement 504
SERVICE RELOAD statement 505
SET statement
  description and format 381
  DOWN BY phrase 382
  index data item values assigned 203
  OFF phrase 383
  ON phrase 383
  overlapping operands, unpredictable results 253
  pointer data item 384
  procedure-pointer data item 385
  requirement for indexed items 166
  SEARCH statement 382
  TO phrase 381
  TO TRUE phrase 383
  UP BY phrase 382
  USAGE IS INDEX clause 203
  USAGE OBJECT REFERENCE 387
sharing data 161
sharing files 145
SHIFT-OUT, SHIFT-IN special registers 17
Sibling program 62
SIGN clause 192
sign condition 237
SIGN IS SEPARATE clause 192
signed
  numeric item, definition 177
  operational signs 139
simple condition
  combined 240
  description and types 225
  negated 239
Simple data reference 48
simple insertion editing 181
SIN function 471
size-error condition 251
skip to next page 40
SKIP1/2/3 statement 506
slack bytes
  between 198
  within 195
slash (/)
  comment line 39
  insertion character 181
  symbol in PICTURE clause 172
SOMClass, root for metaclasses 78
SOMObject, root for classes 78
SOR-FILE-SIZE special register 20
Sort File Description (SD) entry
  Data Division 144
  DATA RECORDS clause 151
  description 140, 144
  level indicator 133
SORT statement
  ASCENDING KEY phrase 389
  COLLATING SEQUENCE phrase 390
  DESCENDING KEY phrase 389
  description and format 388
  DUPLICATES phrase 390
  GIVING phrase 392
  INPUT PROCEDURE phrase 392
  OUTPUT PROCEDURE phrase 393
  USING phrase 391
SORT-CONTROL special register 18
SORT-CORE-SIZE special register 19
SORT-MESSAGE special register 20
SORT-MODE-SIZE special register 21
SORT-RETURN special register 21
Sort/Merge feature
  I-O-CONTROL paragraph format 121
  MERGE statement 327
  RELEASE statement 366
  RERUN clause 123
  RETURN statement 368
  SAME SORT AREA clause 125

Sort/Merge feature *(continued)*
SAME SORT-MERGE AREA clause 125
SORT statement 388
Sort/Merge file statement phrases
ASCENDING/DESCENDING KEY phrase 328
COLLATING SEQUENCE phrase 329
GIVING phrase 330
OUTPUT PROCEDURE phrase 331
USING phrase 330
source code listing 485
source language debugging 529
source program
library, programming notes 491
standard COBOL reference format 34
SOURCE-COMPUTER paragraph 85
special insertion editing 181
special registers
ADDRESS OF 13
DEBUG-ITEM 13
LENGTH OF 15
LINAGE-COUNTER 16
RETURN CODE 17
SHIFT-OUT, SHIFT-IN 17
SORT-CONTROL 18
SORT-CORE-SIZE 19
SORT-FILE-SIZE 20
SORT-MESSAGE 20
SORT-MODE-SIZE 21
SORT-RETURN 21
TALLY 22
WHEN-COMPILED 22
SPECIAL-NAMES paragraph
ACCEPT statement 262
ALPHABET clause 91
ASCII-encoded file specification 156
CLASS clause 94
CODE-SET clause and 156
CURRENCY SIGN clause 95
DECIMAL-POINT IS COMMA clause 96
description 88
format 89
mnemonic names 90
specifications 546
SQRT function 472
standard alignment
JUSTIFIED clause 162
rules 137
standard COBOL format 34
standard data format 138
STANDARD-1 phrase 91
STANDARD-2 phrase 91
STANDARD-DEVIATION function 473
standards 546
START statement
affect on record locking 118
description and format 395
indexed file 396
INVALID KEY phrase 258, 396
relative file 397
status key considerations 396
statement
categories of 245
conditional 246
data manipulation 253
delimited scope 247
description 33, 222
imperative 245
input-output 254
procedure branching 260
statement operations
common phrases 249
file position indicator 260
INTO/FROM identifier phrase 259
status key
common processing facility 254
file processing 509
value and meaning 255
STOP RUN statement 398
STOP statement 398
storage
map listing 485
MEMORY SIZE clause 86
REDEFINES clause 186
STRING statement
description and format 400
execution of 402
overlapping operands, unpredictable results 253
structure of the COBOL language 4
structured programming
DO-WHILE and DO-UNTIL 349
subclasses and methods 78
subjects in EVALUATE statement 295
subprogram linkage
CALL statement 270
CANCEL statement 277
ENTRY statement 294
subprogram termination
CANCEL statement 277
EXIT PROGRAM statement 301

subprogram termination *(continued)*
GOBACK statement 302
subscripting
definition and format 51
INDEXED BY phrase of OCCURS clause 166
MOVE statement evaluation 333
OCCURS clause specification 163
table references 51
using data-names 53
using index-names (indexing) 53
using integers 53
substitution field of INSPECT REPLACING 312
SUBTRACT statement
common phrases 249
description and format 405
SUM function 474
SUPER special character word 9, 319
SUPPRESS option, COPY 488
suppress output 484
suppression editing 184
switch-status condition 238
SYMBOLIC CHARACTERS clause 93
symbols in PICTURE clause 170
SYNCHRONIZED clause 193
VALUE clause and 208
syntax notation, rules for xiii
system considerations, subprogram linkage
CALL statement 270
CANCEL statement 277
system information transfer, ACCEPT statement 263
system input device, ACCEPT statement 262
system-name 86
computer-name 85
SOURCE-COMPUTER paragraph 85

# T

table references
indexing 53
subscripting 51
TALLY special register 22
TALLYING phrase
INSPECT statement 312
UNSTRING statement 410
TAN function 475
termination of execution
EXIT METHOD statement 300
EXIT PROGRAM statement 301
GOBACK statement 302
STOP RUN statement 398
terminators, scope 248
text words 487
text-name
literal-1 486
under AIX and OS/2 42
THREAD compiler option 166
requirement for indexed items 166
THROUGH (THRU) phrase
ALPHABET clause 92
CLASS clause 95
EVALUATE statement 296
PERFORM statement 346
RENAMES clause 190
VALUE clause 209
TIME 265
TIMES phrase of PERFORM statement 348
TITLE statement 507
TO phrase, SET statement 381
TO TRUE phrase, SET statement 383
transfer of control
ALTER statement 269
explicit 58
GO TO statement 303
IF statement 306
implicit 58
PERFORM statement 346
transfer of data
ACCEPT statement 261
MOVE statement 333
STRING statement 400
UNSTRING statement 407
truncation of data
arithmetic item 138
JUSTIFIED clause 162
ROUNDED phrase 250
TRUNC compiler option 139
truth value
complex conditions 238
EVALUATE statement 296
IF statement 305
of complex condition 239
sign condition 238
with conditional statement 246
type conformance
general rules 80
INVOKE...USING 323
multiple inheritance 78
SET...USAGE OBJECT REFERENCE 387

# U

unary operator 224
unconditional GO TO statement 303
unit file, definition 280
UNIT phrase 279
universal object reference 203
unsigned numeric item, definition 177
UNSTRING statement
  description and format 407
  execution 412
  overlapping operands, unpredictable results 253
  receiving field 409
  sending field 407
UP BY phrase, SET statement 382
UPON phrase, DISPLAY 289
UPPER-CASE function 476
UPSI-0 through UPSI-7, program switches
  and switch-status condition 238
  condition-name 91
  processing special conditions 90
  SPECIAL-NAMES paragraph 90
USAGE clause
  BINARY phrase 200
  CODE-SET clause and 156
  COMPUTATIONAL phrases 201
  description 200
  DISPLAY phrase 202
  DISPLAY-1 phrase 202
  elementary item size 138
  format 200
  INDEX phrase 203
  operational signs and 139
  PACKED-DECIMAL phrase 201
  USAGE IS PROCEDURE-POINTER 206
  VALUE clause and 208
USAGE DISPLAY
  class condition identifier 226
  STRING statement and 400
USAGE IS COMPUTATIONAL phrases 201
USAGE IS OBJECT REFERENCE syntax 200
USAGE IS POINTER 205
USAGE IS PROCEDURE-POINTER 206
USAGE OBJECT REFERENCE phrase 319
USE...AFTER...LABEL PROCEDURE on AIX and OS/2 510
user labels
  DEBUGGING declarative 512
  LABEL declarative 510
USING phrase
USING phrase *(continued)*
  CALL statement 271
  in Procedure Division header 217
  MERGE statement 330
  on INVOKE statement 320
  SORT statement 391
  subprogram linkage 219

# V

V, symbol in PICTURE clause 172
VALUE clause
  condition-name 209
  effect on object-oriented programs 129
  format 207, 209
  level 88 item 136
  NULL 211
  rules for condition-name values 210
  rules for literal values 208
VALUE OF clause
  description 151
  format 140
variable-length tables 166
VARIANCE function 477
VARYING phrase
  PERFORM statement 350
  SEARCH statement 376

# W

WHEN phrase
  EVALUATE statement 295
  SEARCH statement 375
WHEN-COMPILED function 478
WHEN-COMPILED special register 22
Windows COBOL language differences
  ACCEPT statement 261
  APPLY WRITE-ONLY 126
  ASSIGN clause 104
  basis-names, library-names, text-names 42
  BLOCK CONTAINS 145
  CLOSE statement 279
  CODE-SET 156
  COMP-5 data item 202
  control characters 138
  environment-name 263
  file handling 126
  file status data-name-8 119
  LABEL RECORDS 150
  LINE SEQUENTIAL file I-O 110

Windows COBOL language differences *(continued)*
LOCK MODE clause 117
MORE-LABELS, GO TO 304
pointer data item size 205
RECORD CONTAINS clause 148
RECORDING MODE clause 154
RERUN clause 122
RESERVE clause 107
SAME AREA clause 123
SAME SORT AREA clause 125
SAME SORT-MERGE AREA clause 125
SET statement 384
SORT-CONTROL special register 18
SORT-FILE-SIZE under AIX and OS/2 20
SORT-MESSAGE under AIX and OS/2 20
SORT-MODE-SIZE under AIX and OS/2 21
status key value and meaning 255
USE...AFTER...LABEL PROCEDURE 510
WRITE ADVANCING 90, 417
WITH DEBUGGING MODE clause 85, 529
WITH DUPLICATES phrase, SORT statement 390
WITH FOOTING phrase 152
WITH NO ADVANCING phrase 289
WITH NO REWIND phrase, CLOSE statement 280
WITH POINTER phrase
STRING statement 401
UNSTRING statement 410
working storage, levels under AIX and OS/2 384
Working-Storage 129
workstation COBOL language differences
ACCEPT statement 261
APPLY WRITE-ONLY 126
ASSIGN clause 104
basis-names, library-names, text-names 42
BLOCK CONTAINS 145
CLOSE statement 279
CODE-SET 156
COMP-5 data item 202
compiler limits 518
control characters 138
DBCS 4
environment-name 263
file handling 126
file status data-name-8 119
LABEL RECORDS 150
LINE SEQUENTIAL file I-O 110
MORE-LABELS, GO TO 304
pointer data item size 205
RECORD CONTAINS clause 148
RECORDING MODE clause 154
workstation COBOL language differences *(continued)*
RERUN clause 122
RESERVE clause 107
SAME AREA clause 123
SAME SORT AREA clause 125
SAME SORT-MERGE AREA clause 125
SET statement 384
SORT-CONTROL special register 18
SORT-FILE-SIZE under AIX and OS/2 20
SORT-MESSAGE under AIX and OS/2 20
SORT-MODE-SIZE under AIX and OS/2 21
status key value and meaning 255
USE...AFTER...LABEL PROCEDURE 510
WRITE ADVANCING 90, 417
WRITE
ADVANCING under AIX and OS/2 90, 417
WRITE statement
affect on record locking 118
AFTER ADVANCING 417, 422
ALTERNATE RECORD KEY 424
BEFORE ADVANCING 417, 422
description and format 415
END-OF-PAGE phrases 419
FROM identifier phrase 259
sequential files 417

# X

X 172
X'00' - X'1F' control characters 138

# Z

Z
insertion character 184
symbol in PICTURE clause 172
zero
filling, elementary moves 334
suppression and replacement editing 184
ZERO in sign condition 238

# We'd Like to Hear from You

Language Reference

IBM COBOL Set for AIX
IBM COBOL for MVS & VM
IBM VisualAge for COBOL for OS/2

Publication No. SC26-4769-03

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773.
- Electronic mail—Use one of the following network IDs:
  - IBMMail: USIB2VVG at IBMMAIL
  - IBMLink: COBPUBS at STLVM27
  - Internet: COMMENTS@VNET.IBM.COM

  Be sure to include the following with your comments:
  - Title and publication number of this book
  - Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

# Readers' Comments

**Language Reference**

**IBM COBOL Set for AIX**
**IBM COBOL for MVS & VM**
**IBM VisualAge for COBOL for OS/2**

**Publication No. SC26-4769-03**

How satisfied are you with the information in this book?

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|---|---|---|---|---|---|
| Technically accurate | ☐ | ☐ | ☐ | ☐ | ☐ |
| Complete | ☐ | ☐ | ☐ | ☐ | ☐ |
| Easy to find | ☐ | ☐ | ☐ | ☐ | ☐ |
| Easy to understand | ☐ | ☐ | ☐ | ☐ | ☐ |
| Well organized | ☐ | ☐ | ☐ | ☐ | ☐ |
| Applicable to your tasks | ☐ | ☐ | ☐ | ☐ | ☐ |
| Grammatically correct and consistent | ☐ | ☐ | ☐ | ☐ | ☐ |
| Graphically well designed | ☐ | ☐ | ☐ | ☐ | ☐ |
| Overall satisfaction | ☐ | ☐ | ☐ | ☐ | ☐ |

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name _____ Address _____

Company or Organization _____ _____

Phone No. _____ _____

**Readers' Comments**
SC26-4769-03

IBM®

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department W92/H3
PO Box 49023
San Jose, CA 95161-9945

Fold and Tape **Please do not staple** Fold and Tape

SC26-4769-03

Cut or Fold
Along Line

IBM®

Printed in U.S.A.

SC26-4769-03