COBOL for OS/390® & VM
COBOL Set for AIX®
VisualAge® COBOL

IBM

Language Reference

SC26-9046-03

Note!

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

Fourth Edition (November 1998)

This edition applies to:

IBM COBOL for OS/390 & VM Version 2 Release 1 Modification 1 (Program Number 5648-A25)
IBM COBOL Set for AIX Release 1 (Program Number 5765-548)
IBM VisualAge COBOL Version 2.2 (Program Number 5639-B92)

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

This edition also applies to:

IBM COBOL for MVS® & VM Version 1 Release 2 Modification 2 (Program Number 5688-197)

When using this edition for IBM COBOL for MVS & VM, treat references to OS/390 as if they were references to MVS.

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

Editions marked “softcopy only” cannot be ordered as printed publications. For information about obtaining these editions, see “Softcopy Publications for IBM COBOL” on page 576.

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, HHX/H3
P.O. Box 49023
San Jose, CA 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, 1998. 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
Fourth Edition (November 1998) xvii
Third Edition (August 1998, Softcopy Only) xvii
Second Edition (April 1998) xvii

Part 1. COBOL Language Structure 1

Characters 2
Character-Strings 3
Figurative Constants 8
Special Registers 10
Literals 20
Separators 28

Sections and Paragraphs 30
Statements and Clauses 30

Reference Format 32
Sequence Number Area 32
Indicator Area 32
Area A 33
Area B 34
Area A or Area B 37

Scope of Names 39
Types of Names 39
External and Internal Resources 42
Resolution of Names 43

Referencing Data Names, Copy Libraries, and Procedure Division Names . 44
Uniqueness of Reference 44

Transfer of Control 56

© Copyright IBM Corp. 1991, 1998

Millennium Language Extensions and Date Fields 58
Millennium Language Extensions Syntax 58
Terms and Concepts 59

Part 2. COBOL Source Unit Structure 63

COBOL Program Structure 64
Nested Programs 66

COBOL Class Definition Structure 69

COBOL Method Definition Structure 71

Part 3. Identification Division 73

Identification Division 74
PROGRAM-ID Paragraph 77
CLASS-ID Paragraph 79
METHOD-ID Paragraph 81
Optional Paragraphs 83

Part 4. Environment Division 85

Configuration Section 86
SOURCE-COMPUTER Paragraph 87
OBJECT-COMPUTER Paragraph 88
SPECIAL-NAMES Paragraph 89
ALPHABET Clause 92
SYMBOLIC CHARACTERS Clause 95
CLASS Clause 95
CURRENCY SIGN Clause 96
REPOSITORY Paragraph 98

Input-Output Section 100
FILE-CONTROL Paragraph 102
SELECT Clause 106
ASSIGN Clause 106
RESERVE Clause 110
ORGANIZATION Clause 111
PADDING CHARACTER Clause 114
RECORD DELIMITER Clause 114
ACCESS MODE Clause 115
RECORD KEY Clause 117
ALTERNATE RECORD KEY Clause 118
RELATIVE KEY Clause 119
PASSWORD Clause 120

LOCK MODE Clause (OS/2 VSAM Files Only) . . . 120
FILE STATUS Clause . . . 122
I-O-CONTROL Paragraph . . . 124
RERUN Clause . . . 125
SAME AREA Clause . . . 127
SAME RECORD AREA Clause . . . 127
SAME SORT AREA Clause . . . 128
SAME SORT-MERGE AREA Clause . . . 129
MULTIPLE FILE TAPE Clause . . . 129
APPLY WRITE-ONLY Clause . . . 129

Part 5. Data Division . . . 131

Data Division Overview . . . 132
File Section . . . 133
Working-Storage Section . . . 133
Local-Storage Section . . . 135
Linkage Section . . . 135
Data Types . . . 136
Data Relationships . . . 137

Data Division—File Description Entries . . . 144
File Section . . . 147
EXTERNAL Clause . . . 148
GLOBAL Clause . . . 149
BLOCK CONTAINS Clause . . . 149
RECORD Clause . . . 151
LABEL RECORDS Clause . . . 154
VALUE OF Clause . . . 155
DATA RECORDS Clause . . . 155
LINAGE Clause . . . 155
RECORDING MODE Clause . . . 157
CODE-SET Clause . . . 159

Data Division—Data Description Entry . . . 161
Format 1 . . . 161
Format 2 . . . 162
Format 3 . . . 162
Level-Numbers . . . 162
BLANK WHEN ZERO Clause . . . 164
DATE FORMAT Clause . . . 164
EXTERNAL Clause . . . 170
GLOBAL Clause . . . 170
JUSTIFIED Clause . . . 171
OCCURS Clause . . . 172
PICTURE Clause . . . 178
REDEFINES Clause . . . 195
RENAMES Clause . . . 198

SIGN Clause 200
SYNCHRONIZED Clause 202
USAGE Clause 209
VALUE Clause 217

Part 6. Procedure Division 223

Procedure Division Structure 225
Requirements for a Method Procedure Division 226
The Procedure Division Header 227
Declaratives 230
Procedures 231
Arithmetic Expressions 233
Conditional Expressions 239
Statement Categories 261
Statement Operations 264

Procedure Division Statements 277
ACCEPT Statement 277
ADD Statement 282
ALTER Statement 285
CALL Statement 287
CANCEL Statement 294
CLOSE Statement 296
COMPUTE Statement 300
CONTINUE Statement 302
DELETE Statement 303
DISPLAY Statement 305
DIVIDE Statement 309
ENTRY Statement 312
EVALUATE Statement 313
EXIT Statement 317
EXIT METHOD Statement 318
EXIT PROGRAM Statement 319
GOBACK Statement 320
GO TO Statement 321
IF Statement 323
INITIALIZE Statement 325
INSPECT Statement 328
INVOKE Statement 337
MERGE Statement 345
MOVE Statement 352
MULTIPLY Statement 357
OPEN Statement 359
PERFORM Statement 365
READ Statement 376
RELEASE Statement 385
RETURN Statement 387

REWRITE Statement . . . 389
SEARCH Statement . . . 393
SET Statement . . . 400
SORT Statement . . . 407
START Statement . . . 415
STOP Statement . . . 418
STRING Statement . . . 420
SUBTRACT Statement . . . 425
UNSTRING Statement . . . 428
WRITE Statement . . . 436

Part 7. Intrinsic Functions . . . 445

Intrinsic Functions . . . 447
Specifying a Function . . . 447
Function Definitions . . . 454
ACOS . . . 458
ANNUITY . . . 459
ASIN . . . 460
ATAN . . . 461
CHAR . . . 462
COS . . . 463
CURRENT-DATE . . . 464
DATE-OF-INTEGER . . . 466
DATE-TO-YYYYMMDD . . . 467
DATEVAL . . . 468
DAY-OF-INTEGER . . . 470
DAY-TO-YYYYDDD . . . 471
FACTORIAL . . . 472
INTEGER . . . 473
INTEGER-OF-DATE . . . 474
INTEGER-OF-DAY . . . 475
INTEGER-PART . . . 476
LENGTH . . . 477
LOG . . . 478
LOG10 . . . 479
LOWER-CASE . . . 480
MAX . . . 481
MEAN . . . 482
MEDIAN . . . 483
MIDRANGE . . . 484
MIN . . . 485
MOD . . . 486
NUMVAL . . . 487
NUMVAL-C . . . 488
ORD . . . 490
ORD-MAX . . . 491
ORD-MIN . . . 492

PRESENT-VALUE . . . 493
RANDOM . . . 494
RANGE . . . 495
REM . . . 496
REVERSE . . . 497
SIN . . . 498
SQRT . . . 499
STANDARD-DEVIATION . . . 500
SUM . . . 501
TAN . . . 502
UNDATE . . . 503
UPPER-CASE . . . 504
VARIANCE . . . 505
WHEN-COMPILED . . . 506
YEAR-TO-YYYY . . . 508
YEARWINDOW . . . 509

Part 8. Compiler-Directing Statements . . . 511

Compiler-Directing Statement . . . 512
BASIS Statement . . . 512
CBL (PROCESS) Statement . . . 513
*CONTROL (*CBL) Statement . . . 514
COPY Statement . . . 516
DELETE Statement . . . 523
EJECT Statement . . . 524
ENTER Statement . . . 524
INSERT Statement . . . 525
READY or RESET TRACE Statement . . . 526
REPLACE Statement . . . 527
SERVICE LABEL Statement . . . 530
SERVICE RELOAD Statement . . . 531
SKIP1/2/3 Statements . . . 531
TITLE Statement . . . 532
USE Statement . . . 533

Compiler Directives . . . 539
CALLINTERFACE . . . 539

Appendixes . . . 543

Appendix A. Compiler Limits . . . 544

Appendix B. EBCDIC and ASCII Collating Sequences . . . 548
EBCDIC Collating Sequence . . . 548
US English ASCII Code Page (ISO 646) . . . 551

Appendix C. Source Language Debugging 555
Coding Debugging Lines 555
Coding Debugging Sections 555
DEBUG-ITEM Special Register 556
Activate Compile-Time Switch 556
Activate Object-Time Switch 556

Appendix D. Reserved Words 558

Appendix E. ASCII Considerations for OS/390 and VM 565
Environment Division 565
Data Division 567
Procedure Division 567

Appendix F. Locale Considerations (Workstation Only) 568

Appendix G. Summary of Language Difference: Host COBOL and Workstation COBOL 569

Appendix H. Industry Specifications 571
Standard Terminology 573

Bibliography 574

Glossary 577

Index 600

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
North Castle Drive
Armonk, NY 10504-1785
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, HHX/H3
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

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

Programming Interface Information

This *Language Reference* documents intended Programming Interfaces that allow the customer to write programs to obtain the services of COBOL for OS/390 & VM, COBOL Set for AIX, and VisualAge COBOL.

© Copyright IBM Corp. 1991, 1998

Notices

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

Advanced Function Printing
AFP
AIX
AIX/6000
BookManager
CICS
CICS/ESA
DATABASE 2
DB2
DFSORT
IBM
IMS/ESA
MVS
Operating System/2
OS/2
OS/390
Print Services Facility
SOM
SOMobjects
VisualAge
VM/ESA

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

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

Other company, product, and service names may be trademarks or service marks of others.

About This Book

This book presents the syntax of COBOL for OS/390 & VM, COBOL Set for AIX, and VisualAge COBOL (collectively referred to in this book as IBM COBOL). To indicate platform-specific information, this book use the following methods:

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. This book uses the following icons:

Host Informs you of information specific to COBOL for OS/390 & VM. Host

Workstation Informs you of information specific to COBOL Set for AIX and VisualAge COBOL (under OS/2® and Windows). Workstation

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

OS/2 Informs you of information specific to VisualAge COBOL, under OS/2 only. OS/2

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. In the hard-copy, published book, IBM extensions appear in gray ink. For example:

IBM extensions in text are shown this way.

IBM extensions are not indicated in the 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.)

© Copyright IBM Corp. 1991, 1998

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

ALTER statement
AUTHOR paragraph
Comment entry
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.

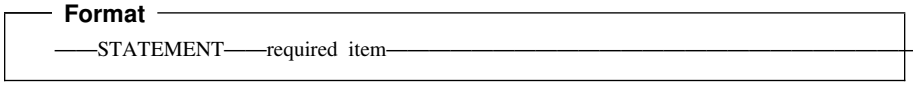
The ——— symbol indicates that the syntax diagram is continued from the previous line.

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.

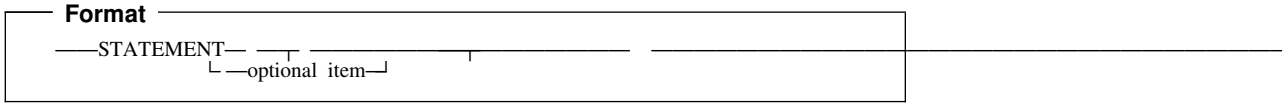
Required items appear on the horizontal line (the main path).

Format



Optional items appear below the main path.

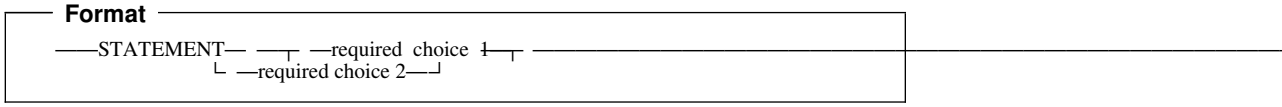
Format



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

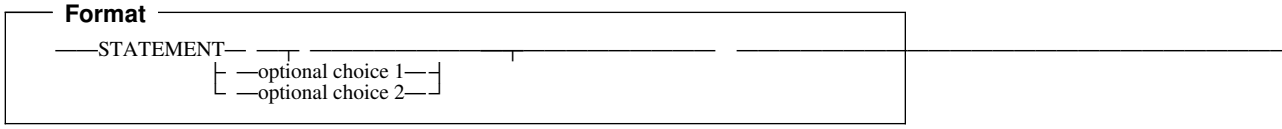
If you **must** choose one of the items, one item of the stack appears on the main path.

Format



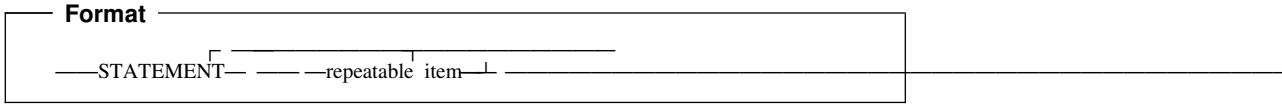
If choosing one of the items is optional, the entire stack appears below the main path.

Format



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

Format



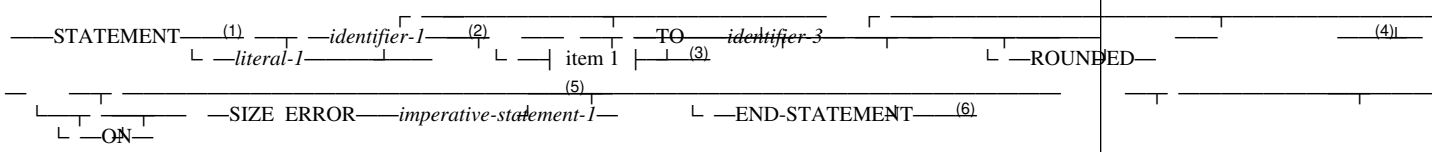
A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

Variables appear in all lowercase letters (for example, parmx). They represent user-supplied names or values.

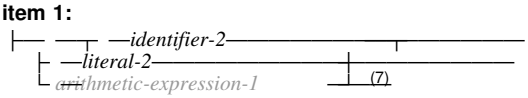
If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, they must be entered as part of the syntax.

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 The gray text indicates that arithmetic-expression-1 is an IBM extension. This operand is optional.

DBCS Notation

Host 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'. Host

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

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

Major changes to the COBOL for OS/390 & VM, COBOL Set for AIX, and VisualAge COBOL languages are listed below, according to the edition in which they first appeared. Changes to the language since the previous edition of this book are marked by a vertical bar in the left margin.

Fourth Edition (November 1998)

Enhancements to the millennium language extensions:

- Additional date patterns for the DATE FORMAT clause, including "year-last" dates.
- DATE FORMAT for binary numeric items.
- Relaxation of the USING/RETURNING parameter rules for windowed date fields.
- Host Special semantics for "trigger" and "limit" date values. For more information, see "Semantics of Windowed Date Fields" on page 165.

New sub-option TRIG/NOTRIG of the DATEPROC compiler option, to enable or disable trigger and limit processing. Host

Third Edition (August 1998, Softcopy Only)

Extensions to support the Euro currency sign in numeric-edited data items. These extensions introduce a PICTURE SYMBOL phrase to the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. The PICTURE SYMBOL phrase allows a PICTURE clause *currency symbol* to represent a *currency sign value* that is different from the currency symbol, and not restricted to a single character. For example, the currency symbol '$' could be used to represent a code point for the Euro currency sign, or the characters 'EUR'. The extensions also allow multiple currency symbols and currency sign values to be defined. For details, see "CURRENCY SIGN Clause" on page 96.

Update to the millennium language extensions to allow signed numeric date fields.

Second Edition (April 1998)

The millennium language extensions, enabling compiler-assisted date processing for dates containing 2-digit and 4-digit years.

Host Requires IBM VisualAge COBOL Millennium Language Extensions for OS/390 & VM (program number 5648-MLE) to be installed with your compiler. Host

For information on the millennium language extensions, see "Millennium Language Extensions and Date Fields" on page 58.

New language elements in support of the millennium language extensions:

© Copyright IBM Corp. 1991, 1998

- DATE FORMAT clause in data description entries
- Intrinsic functions:
  - DATEVAL
  - UNDATE
  - YEARWINDOW

New compiler options in support of the millennium language extensions:

- DATEPROC/NODATEPROC
- YEARWINDOW

New compiler option, ANALYZE, to check the syntax of imbedded SQL and CICS® statements.

New date intrinsic functions to cover the recommendation in the *Working Draft for Proposed Revision of ISO 1989:1985 Programming Language COBOL*:

- DATE-TO-YYYYMMDD
- DAY-TO-YYYYDDD
- YEAR-TO-YYYY

Extension of the ACCEPT statement to cover the recommendation in the *Working Draft for Proposed Revision of ISO 1989:1985 Programming Language COBOL*:

- ACCEPT FROM DATE YYYYMMDD
- ACCEPT FROM DAY YYYYDDD

Part 1. COBOL Language Structure

Characters 2
Character-Strings 3
Figurative Constants 8
Special Registers 10
Literals 20
Separators 28

Sections and Paragraphs 30
Statements and Clauses 30

Reference Format 32
Sequence Number Area 32
Indicator Area 32
Area A 33
Area B 34
Area A or Area B 37

Scope of Names 39
Types of Names 39
External and Internal Resources 42
Resolution of Names 43

Referencing Data Names, Copy Libraries, and Procedure Division Names . 44
Uniqueness of Reference 44

Transfer of Control 56

Millennium Language Extensions and Date Fields 58
Millennium Language Extensions Syntax 58
Terms and Concepts 59

© Copyright IBM Corp. 1991, 1998

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 on page 3.

The basic IBM COBOL language is restricted to the character set shown in Table 1 on page 3, 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.

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

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

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

DBCS and EUC characters are valid characters in certain COBOL character-strings. For details, see “COBOL Words with Multi-Byte Characters” on page 4 and “DBCS Literals” on page 25.

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 28.

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 38.

© Copyright IBM Corp. 1991, 1998

Table 1. Characters—Meanings

Character	Meaning
	Space
+	Plus sign
–	Minus sign or Hyphen
*	Asterisk
/	Slant, Solidus, Stroke, or Slash
=	Equal sign
$	Currency sign
,	Comma
;	Semicolon
.	Decimal point or Period
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than
<	Less than
:	Colon
'	Apostrophe
A–Z	Alphabet (uppercase)
a–z	Alphabet (lowercase)
0–9	Numeric characters

Character-Strings

You can use EBCDIC and/or DBCS character strings under OS/390 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 lower-

case 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 (Page 1 of 2). Rules for Forming User-Defined Words from Multi-Byte Characters

Rule	OS/390 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.	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. 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.	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

Table 2 (Page 2 of 2). Rules for Forming User-Defined Words from Multi-Byte Characters

Rule	OS/390 and VM	AIX, OS/2, and Windows
Maximum Length	14 characters	15 characters for a DBCS code page For AIX only: 7 characters for EUC code page IBM_eucTW 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:

	Multi-Byte Characters Allowed?
Alphabet-name	Yes
Class-name	Yes
Condition-name	Yes
Data-name	Yes
File-name	Yes
Index-name	Yes
Level-numbers: 01–49, 66, 77, 88	Yes
Library-name	No
Method-name	No
Mnemonic-name	Yes
Object-oriented class-name	No
Paragraph-name	Yes
Priority-numbers: 00–99	Yes
Program-name	No
Record-name	Yes
Section-name	Yes
Symbolic-character	Yes
Text-name	No

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 44.

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

Character-Strings

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

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.

Host Under OS/390 and VM, the only DBCS character string system-name allowed is computer-name. Host

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

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 52 on page 455.

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 558.

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

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 8.

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 233.

Relational operators: < > = <= >=

See “Conditional Expressions” on page 239.

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,

Figurative Constants

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 10.

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 568. 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 ("), if the QUOTE compiler option is in effect
or
The apostrophe character ('), if the APOST compiler option is in effect

QUOTE or QUOTES cannot be used in place of a quotation mark or an apostrophe to enclose a nonnumeric literal.

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.

Workstation Under AIX, OS/2, and Windows, you cannot specify the SYMBOLIC CHARACTER clause if a DBCS or EUC code page is indicated by the locale setting. For more information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 568. Workstation

NULL/NULLS
Represents a value used to indicate that data items defined with USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, USAGE IS OBJECT REFERENCE, or the ADDRESS OF special register do not contain a valid address. NULL can be used only where explicitly allowed in the syntax format. NULL has the value of 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 data-name 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 44.)

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
Programs that possess the RECURSIVE attribute
Programs compiled with the THREAD option (Workstation only)

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

In all other cases, the special registers will not be reset; they will be unchanged from 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:

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

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.

Special Registers

DEBUG-CONTENTS

Data is moved into DEBUG-CONTENTS, as shown in Table 3 on page 12.

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 can be 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
```

Note: The number of bytes occupied by a COBOL item is also accessible through the intrinsic function LENGTH (See “LENGTH” on page 477). LENGTH supports nonnumeric 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.

Special Registers

For more information, see “LINAGE Clause” on page 155.

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 436.)

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 a return code to the calling program or operating system when the current COBOL program ends. When a COBOL program ends:

- If control returns to the operating system, the value of the RETURN-CODE special register is passed to the operating system as a user return code. The supported user return code values are determined by the operating system, and might not include the full range of RETURN-CODE special register values. AIX For information on user return code values under AIX, see the *IBM COBOL Set for AIX Programming Guide*. AIX
- If control returns to a calling program, the value of the RETURN-CODE special register is passed to the calling program. If the calling program is a COBOL program, the RETURN-CODE special register in the calling program is set to the value of the RETURN-CODE special register in the called program.

The RETURN-CODE special register has the implicit definition:

```
1  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 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 337 or "CALL Statement" on page 287.

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

The RETURN-CODE special register will not contain return code information:

- On the host, from a service call for a Language Environment callable service. For more information, see the *IBM COBOL for OS/390 & VM Programming Guide* and *Language Environment Programming Guide*.
- On the workstation, from a date/time callable service. For more information, see the *IBM COBOL Programming Guide* for your platform.

SHIFT-OUT and SHIFT-IN

Workstation 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. Workstation

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

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

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:

Special Registers

```
DATA DIVISION.
WORKING-STORAGE.

 1    DBCSGRP.
          5   SO          PIC  X.
          5   DBCSITEM PIC G(3) USAGE DISPLAY-1
          5   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.

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

```
1 SORT-CONTROL GLOBAL PICTURE X(16 ) VALUE "file name".
```

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

Host Under OS/390 and VM it is implicitly defined as:

```
1       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 OS/390, you can provide a DD statement for the data set identified by the SORT-CONTROL special register, and COBOL for OS/390 & VM will attempt to open the data set at execution time. Any error will be diagnosed with an informational message. Host

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:

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

Workstation 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. Workstation

Host Under OS/390 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. Host

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:

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

Workstation 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. Workstation

Host Under OS/390 and CMS, SORT-FILE-SIZE is equivalent to the 'FILSZ=Ennn' control statement in the sort control file. Host

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.

Special Registers

SORT-MESSAGE

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

Workstation 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. Workstation

Host Under OS/390 and CMS, it has the implicit definition:

```
1       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. Host

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:

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

Workstation 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. Workstation

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

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:

```
1       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.

TALLY

The TALLY special register is the name of a binary data item with the following definition:

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

You can refer to or modify the contents of TALLY.

You can specify the TALLY 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.

WHEN-COMPILED

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

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

The WHEN-COMPILED special register has the format:

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

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

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

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

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

Literals

Note: The compilation date and time is also accessible via the date/time intrinsic function WHEN-COMPILED (See “WHEN-COMPILED” on page 506). That function supports 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 8.) The literal types are **nonnumeric**, **DBCS**, and **numeric**.

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"
```

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

```
'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,

```
'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 140.)

Table 4 on page 21 lists when nonnumeric literals with double-byte or multiple-byte characters cannot be used.

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

OS/390 and VM	AIX, OS/2, and Windows
As a literal in the following:	As a literal in the following:
ALPHABET clause ASSIGN clause CALL statement program-id CANCEL statement CLASS clause CURRENCY SIGN clause END METHOD header END PROGRAM header ENTRY statement METHOD-ID paragraph PADDING CHARACTER clause PROGRAM-ID paragraph RERUN clause STOP statement	ALPHABET clause ASSIGN clause CLASS clause CURRENCY SIGN clause END METHOD header METHOD-ID paragraph PADDING CHARACTER clause RERUN clause STOP statement
BASIS statement (basis-name)	
COPY statement (text-name)	
COPY statement (library-name)	

Workstation 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. Workstation

Host Under OS/390 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.

Literals

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.

Under COBOL for OS/390 & VM, COBOL statements process nonnumeric literals with double-byte characters without sensitivity to the shift codes and character codes. The use of statements that operate on a byte-to-byte basis (for example, STRING and UNSTRING) can result in strings that are not valid mixtures of EBCDIC and double-byte characters. You must be certain that the statements use DBCS characters. See *IBM COBOL for OS/390 & VM Programming Guide* for more information on using nonnumeric literals and data items with double-byte characters in statements that operate on a byte-by-byte basis. Host

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

Hexadecimal Notation Format for Nonnumeric Literals

```
X"hexadecimal–digits"
```

X" The opening delimiter for hexadecimal notation of a nonnumeric literal. (Alternatively, you can use apostrophes (') as delimiters.)

'' The closing delimiter for the hexadecimal notation of a nonnumeric literal. (Alternatively, you can use apostrophes (') as delimiters.)

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

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

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

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

The padding character for hexadecimal notation of nonnumeric literals is the blank (X'40' for OS/390 and VM) or (X'20' for AIX, OS/2, and Windows).

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

Format for Null-Terminated Nonnumeric Literals

```
Z"ddddd"
```

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

'' The closing delimiter for a null-terminated notation of a nonnumeric literal. (Alternatively, you can use apostrophes (') as delimiters.)

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

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

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

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

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

Literals

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.

Numeric literals can be 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 1 or 2 digits.

Host Under OS/390 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. Host

Workstation 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 $1.798(10^{308})$ Workstation

Every numeric literal is in the numeric data category. (Data categories are described under “Classes and Categories of Data” on page 140.)

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. Format and Rules for Forming DBCS Literals

Rules	OS/390 and VM	AIX, OS/2, and Windows
Format	G"<D1D2D3>" N"<D1D2D3>"	G"D1D2D3" N"D1D2D3"
G" N"	Opening delimiters. They must be followed immediately by a shift-out control character. 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.
<	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. 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.

Literals

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. (The value of a DBCS space is X'4040'.)

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 180.

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 83.

Comment line (Any division)
This form is described under “Comment Lines” on page 37.

Character-strings that form comments can contain:

Host Under OS/390 and VM, DBCS characters or a combination of DBCS and EBCDIC characters. Host

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

Multiple comment lines containing DBCS/EUC strings are allowed. The embedding of DBCS/EUC characters in a comment line must be done on a line-by-line basis. DBCS/EUC words cannot be continued to a following line. No syntax checking for valid 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
'	Apostrophe
X"	Opening delimiter for a nonnumeric literal
Z"	Opening delimiter for a null-terminated nonnumeric literal
N"	Opening delimiter for a DBCS literal
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.

Workstation The IBM COBOL character set does not include a tab character (ASCII code 9). You cannot use the tab character as a separator in IBM COBOL.
Workstation

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.

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 35).

Apostrophes {'} ... {'}
An opening apostrophe must be immediately preceded by a space or a left parenthesis. A closing apostrophe must be immediately followed by a separator (space, comma, semicolon, period, or right parenthesis). Apostrophes must appear as balanced pairs. They delimit nonnumeric literals, except when the literal is continued (see "Continuation Lines" on page 35).

Pseudo-text delimiters { ==} . . . {== }
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 516.)

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 231.

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, 1998

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 261
"Conditional Statements" on page 262
"Scope of Names" on page 39
"Compiler-Directing Statement" on page 512

Phrases

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

Indicator Area

Reference Format

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

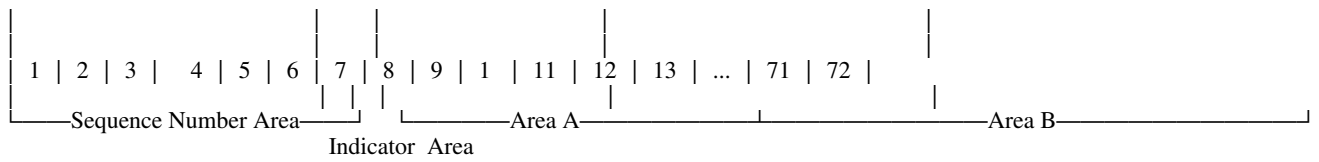


Figure 1. Reference Format for COBOL Source Line

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 35, “Comment Lines” on page 37, and “Debugging Lines” on page 38.

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. For information on the LINECOUNT compiler option, see the *IBM COBOL Programming Guide* for your platform.

© Copyright IBM Corp. 1991, 1998

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 methodheader

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 B

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 147.) 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 230.)

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

Area B

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 30.

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.

DBCS literals and user-defined words containing multi-byte characters cannot be continued.

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

- Hexadecimal notation of a nonnumeric literal (X" or 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.

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

Area B

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..
   1                    "AAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE
     -                    "GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJKKKKKKKKKK
     -      "LLLLLLLLLLMMMMMMMMM"

   5      "AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE
     -      "GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJKKKKKKKKKK
     -      "LLLLLLLLLLMMMMMMMMM"

   1      "AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE"
     -      "GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJKKKKKKKKKK"
     -      "LLLLLLLLLLMMMMMMMMM"
```

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 162.

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.

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

Note: Comments intermixed with control cards could nullify some of the control cards 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.

As an IBM extension BASIS, CBL (PROCESS), *CBL (*CONTROL), DELETE, EJECT, INSERT, SKIP1/2/3, and TITLE can also start in Area A or Area B.

Compiler Directives (Workstation Only)

Compiler directives can start only in Area B. Currently, the only compiler directive is 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 87.

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 170.

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.

For Classes and Methods

Names declared in a class definition are global to all the methods contained in that 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, 1998

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.

method-name
Method-name assigns a name to a method. A method-name is neither local nor 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
Host Under OS/390 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. Host

Workstation 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 516. Workstation

text-name

Host Under OS/390 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. Host

Workstation 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 516.
Workstation

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 methodthat 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 43.

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, 1998

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.

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

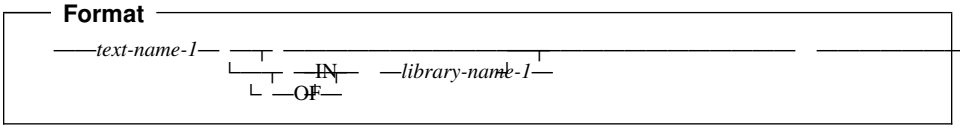
Identical Names

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.

These same rules apply to classes and their contained methods.

References to COPY Libraries

Format



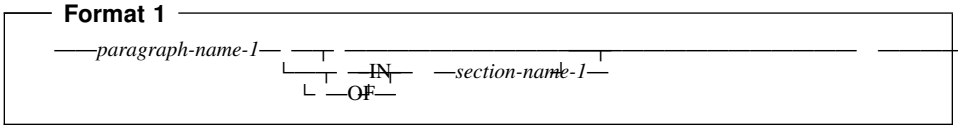
If more than one COBOL library is available to the compiler during compilation, as an IBM extension, text-name-1 need not be qualified each time it is referenced; a qualification of SYSLIB is assumed.

For rules on referencing COPY libraries, see “COPY Statement” on page 516.

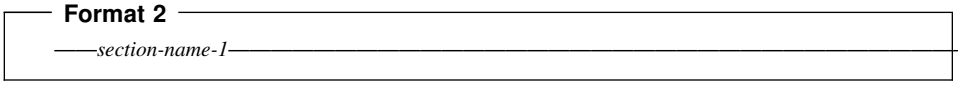
Uniqueness of Reference

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 231, is the highest and only qualifier available for a paragraph-name and must be unique if referenced.

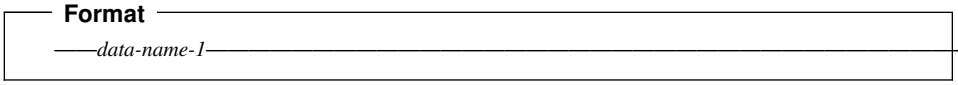
If explicitly referenced, a paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to within the section in which it appears. A paragraph-name or section-name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

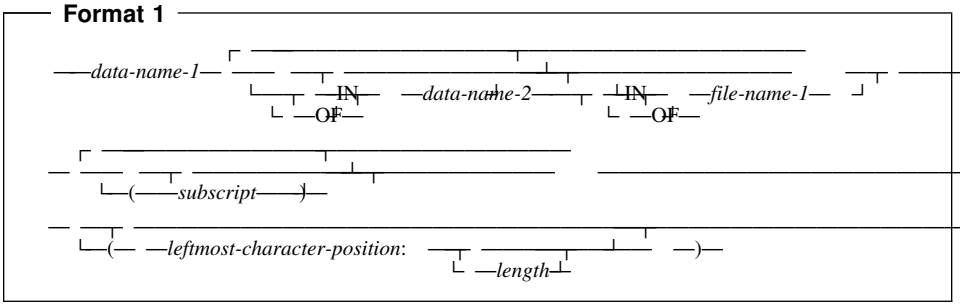
The term **data-name** refers to a name that must not be qualified, subscripted, or reference modified, unless specifically permitted by the rules for the format.

For a description of qualification, see “Qualification” on page 44.

Uniqueness of Reference

For a description of subscripting, see “Subscripting” on page 49.
For a description of reference modification, see “Reference Modification” on page 52.

Format 1

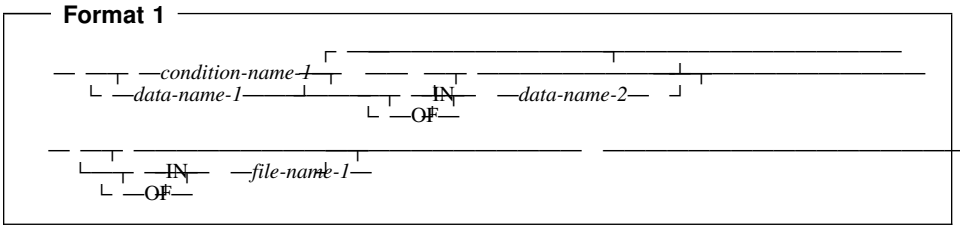


data-name-1, data-name-2
Can be a record-name.

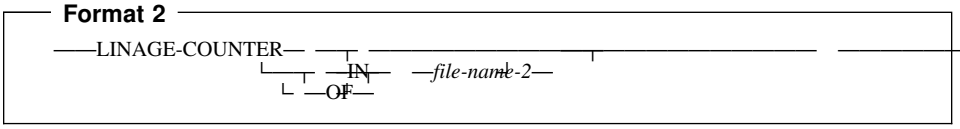
file-name-1
Must be identified by an FD or SD entry in the Data Division.

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.

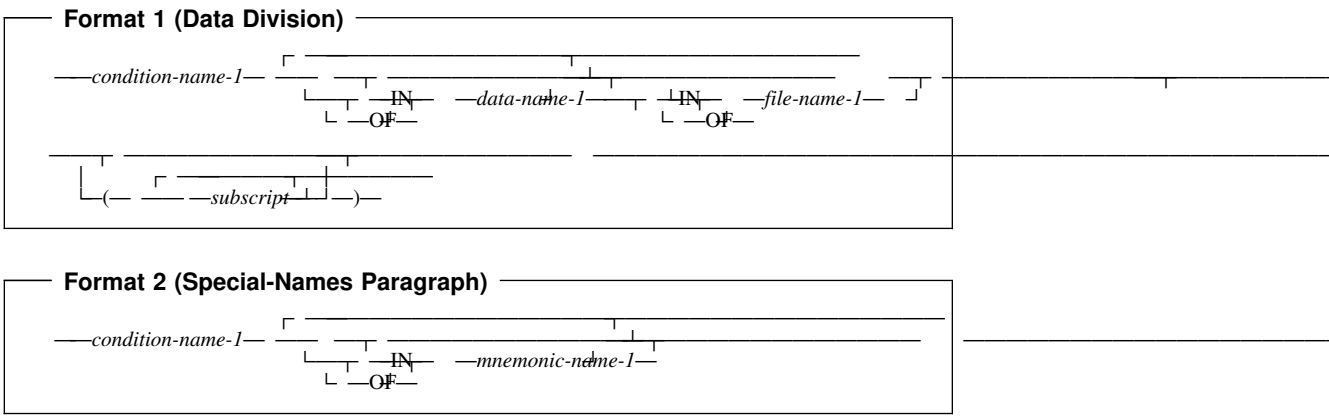
Duplication of data-names must not occur in those places where the data-name cannot be made unique by qualification.

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



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.

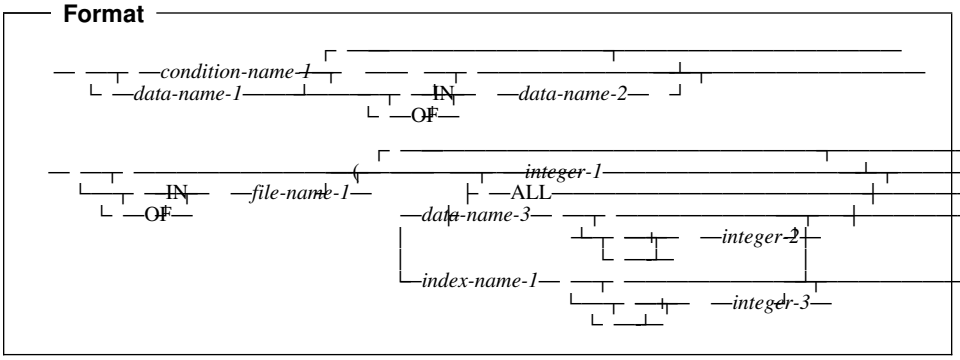
File-name-1 must be unique within this program.

mnemonic-name-1
For information on acceptable values for mnemonic-name-1, see "SPECIAL-NAMES Paragraph" on page 89.

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. Data-name-3 cannot be a windowed date field.

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:

```
1  TABLE-THREE.
     5   ELEMENT-ONE OCCURS 3 TIMES.
         1   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 54. 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

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.

Uniqueness of Reference

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 175.

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.

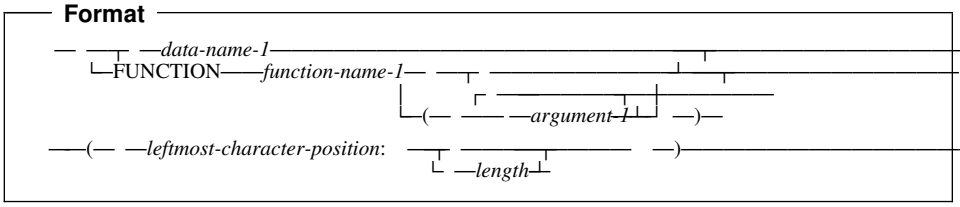
As an IBM extension, the integer can be positively signed.

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 or DISPLAY-1.

Data-name-1 can be qualified or subscripted. Data-name-1 cannot be a windowed date field.

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.

The evaluation of *leftmost-character-position* must not result in a windowed date field.

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.

The evaluation of *length* must not result in a windowed date field.

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

Each character of 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 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.

If data-name-1 is an expanded date field, then the result of reference modification is a non-date.

Reference modification creates a unique data item which is a subset of 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(1 ).
  ⋮

    MOVE WHOLE-NAME(1:1 ) 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.

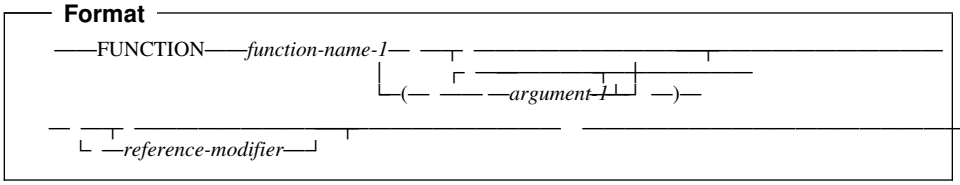
```
 1  TABLE-1.
     2  TAB   OCCURS  1   TIMES   PICTURE  X(5).
77  SUFFIX                        PICTURE  X(2).
  ⋮

    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 447.

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 261.)

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, 1998

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 methodheader

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.

Millennium Language Extensions and Date Fields

Many applications use 2 digits rather than 4 digits to represent the year in date fields, and assume that these values represent years from 1900 to 1999. This compact date format works well for the 1900s, but it does not work for the year 2000 and beyond because these applications interpret “00” as 1900 rather than 2000, producing incorrect results.

The millennium language extensions are designed to allow applications that use 2-digit years to continue performing correctly in the year 2000 and beyond, with minimal modification to existing code. This is achieved using a technique known as windowing, which removes the assumption that all 2-digit year fields represent years from 1900 to 1999. Instead, windowing enables 2-digit year fields to represent years within any 100-year range, known as a **century window**.

For example, if a 2-digit year field contains the value 15, many applications would interpret the year as 1915. However, with a century window of 1960–2059, the year would be interpreted as 2015.

The millennium language extensions provide support for the most common operations on date fields: comparisons, moving and storing, incrementing and decrementing. This support is limited to date fields of certain formats; for details, see “DATE FORMAT Clause” on page 164.

For information on supported operations and restrictions when using date fields, see “Restrictions On Using Date Fields” on page 166.

Millennium Language Extensions Syntax

The millennium language extensions introduce the following language elements to IBM COBOL:

The **DATE FORMAT** clause in data description entries, which defines data items as date fields.

The following intrinsic functions:

DATEVAL	Converts a non-date to a date field.
UNDATE	Converts a date field to a non-date.
YEARWINDOW	Returns the first year of the century window specified by the YEARWINDOW compiler option.

For details on using the millennium language extensions in applications, see the *IBM COBOL Programming Guide* for your platform, or the *IBM COBOL Millennium Language Extensions Guide*.

© Copyright IBM Corp. 1991, 1998

Note: The millennium language extensions have no effect unless:

Host IBM VisualAge COBOL Millennium Language Extensions for OS/390 & VM (program number 5648-MLE) is installed with your compiler. Host

Your COBOL program is compiled using the DATEPROC compiler option (with the century window specified by the YEARWINDOW compiler option).

Terms and Concepts

This book uses the following terms when referring to the millennium language extensions.

Date Field

A **date field** can be any of the following:

A data item whose data description entry includes a DATE FORMAT clause.

A value returned by one of the following intrinsic functions:

DATE-OF-INTEGER
DATE-TO-YYYYMMDD
DATEVAL
DAY-OF-INTEGER
DAY-TO-YYYYDDD
YEAR-TO-YYYY
YEARWINDOW

The conceptual data items DATE, DATE YYYYMMDD, DAY, and DAY YYYYDDD of the ACCEPT statement.

The result of certain arithmetic operations (for details, see “Arithmetic with Date Fields” on page 235).

The term date field refers to both **expanded date fields** and **windowed date fields**.

Windowed Date Field

A windowed date field is a date field that contains a **windowed year**. A windowed year consists of 2 digits, representing a year within the century window.

Expanded Date Field

An expanded date field is a date field that contains an **expanded year**. An expanded year consists of 4 digits.

Note: The main use of expanded date fields is to provide correct results when these are used in combination with windowed date fields; for example, where migration to 4-digit year dates is not complete. If all the dates in an application use 4-digit years, there is no need to use the millennium language extensions.

Year-Last Date Field

A year-last date field is a date field whose DATE FORMAT clause specifies one or more Xs preceding the YY or YYYY. Year-last date fields are supported in a limited number of operations, typically involving another date with the same (year-last) date format, or a non-date.

Date Format

Date format refers to the date pattern of a date field, specified either:

Explicitly, by the DATE FORMAT clause or DATEVAL intrinsic function argument-2

or

Implicitly, by statements and intrinsic functions that return date fields (for details, see “Date Field” on page 59)

Compatible Date Field

The meaning of the term **compatible**, when applied to date fields, depends on the COBOL division in which the usage occurs:

Data Division

Two date fields are compatible if they have identical USAGE and meet at least one of the following conditions:

- They have the same date format
- Both are windowed date fields, where one consists only of a windowed year, DATE FORMAT YY
- Both are expanded date fields, where one consists only of an expanded year, DATE FORMAT YYYY
- One has DATE FORMAT YYXXXX, the other, YYXX
- One has DATE FORMAT YYYYXXXX, the other, YYYYXX

A windowed date field can be subordinate to an expanded date group data item. The two date fields are compatible if the subordinate date field has USAGE DISPLAY, starts two bytes after the start of the group expanded date field, and the two fields meet at least one of the following conditions:

- The subordinate date field has a DATE FORMAT pattern with the same number of Xs as the DATE FORMAT pattern of the group date field.
- The subordinate date field has DATE FORMAT YY.

The group date field has DATE FORMAT YYYYXXXX and the subordinate date field has DATE FORMAT YYXX.

Procedure Division

Two date fields are compatible if they have the same date format except for the year part, which may be windowed or expanded. For example, a windowed date field with DATE FORMAT YYXXX is compatible with:

Another windowed date field with DATE FORMAT YYXXX

An expanded date field with DATE FORMAT YYYYXXX

Non-Date

A **non-date** can be any of the following:

A data item whose date description entry does not include the DATE FORMAT clause

A date field that has been converted using the UNDATE function

A literal

A reference-modified date field

The result of certain arithmetic operations that may include date field operands; for example, the difference between two compatible date fields

Century Window

A century window is a 100-year interval within which any 2-digit year is unique. There are several types of century window available to COBOL programmers:

1. For windowed date fields, it is specified by the YEARWINDOW compiler option
2. For windowing intrinsic functions DATE-TO-YYYYMMDD, DAY-TO-YYYYDDD, and YEAR-TO-YYYY, it is specified by argument-2
3. For Language Environment callable services, it is specified in CEESCEN

Millennium Language Extensions and Date Fields

Part 2. COBOL Source Unit Structure

COBOL Program Structure 64
Nested Programs 66

COBOL Class Definition Structure 69

COBOL Method Definition Structure 71

© Copyright IBM Corp. 1991, 1998

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, see “Nested Programs” on page 66 and the *IBM COBOL 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:

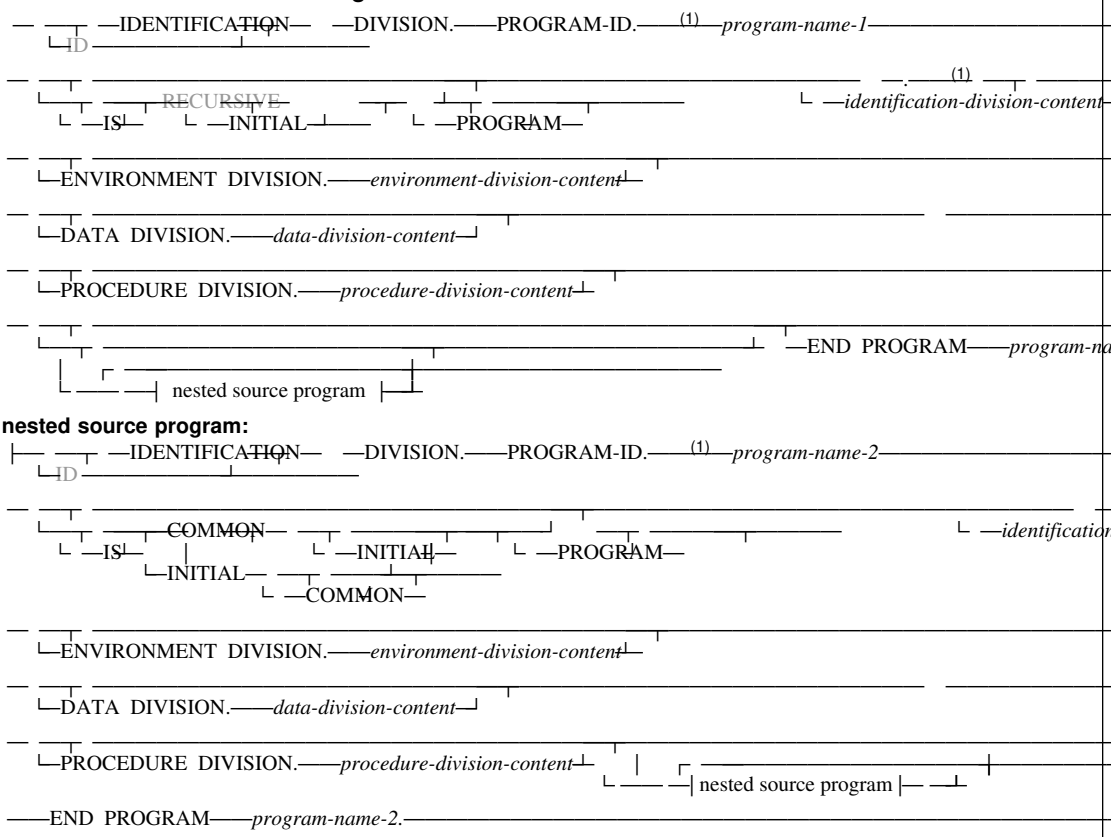
Identification Division
Environment Division
Data Division
Procedure Division

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.

© Copyright IBM Corp. 1991, 1998

Format—COBOL Source Program

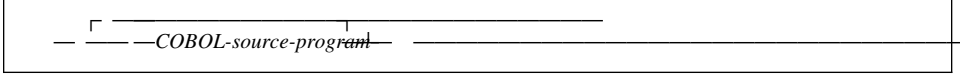


Note:

1 This separator period is optional as an IBM extension.

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.

Program-name can be a nonnumeric literal, but cannot be a figurative constant. The content of the literal must follow the rules for formation of program names. 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.

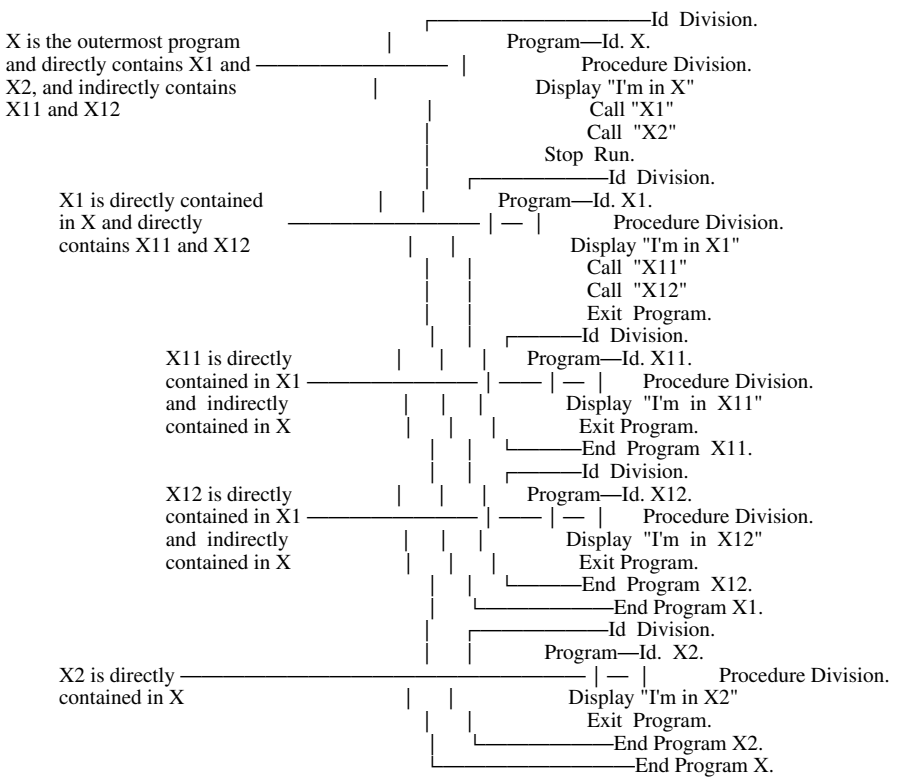


Figure 2. Nested program structure with directly and indirectly contained programs

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.

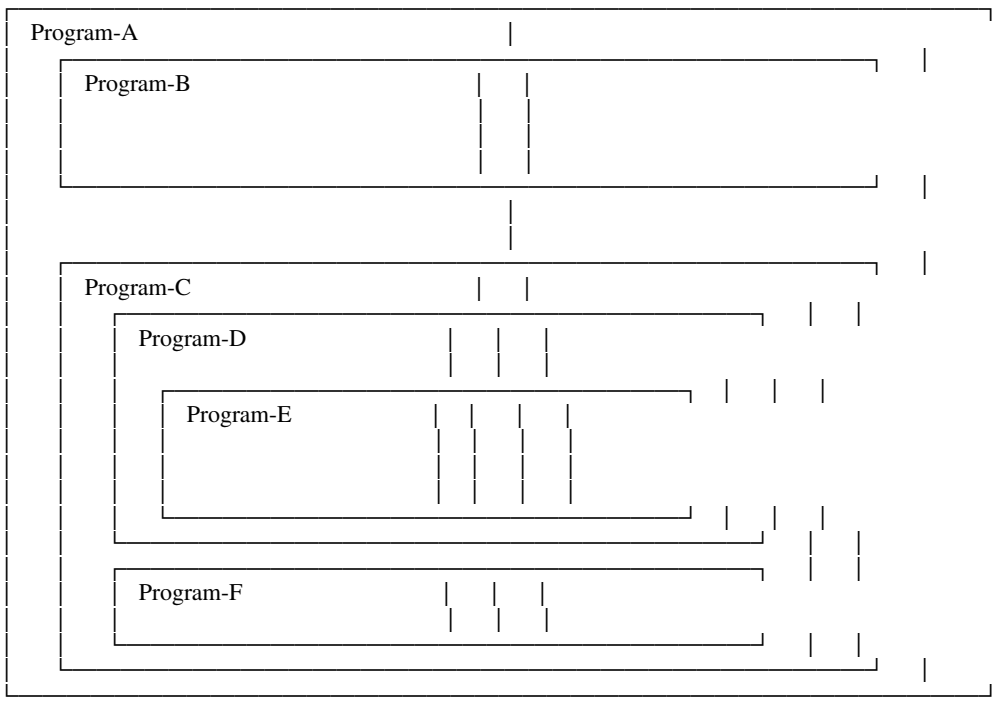
If the program-name is that of a program which is separately compiled, that program-name can be referenced by statements included in any other program in the run unit, except programs it directly or indirectly contains.

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.

COBOL Program Structure



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

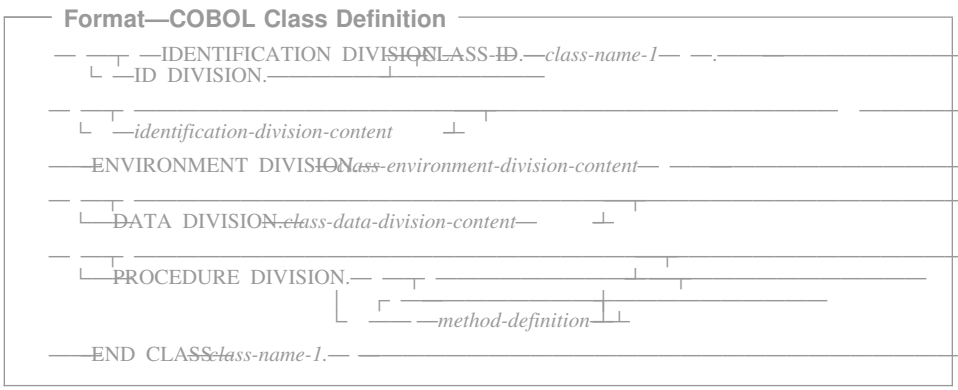
The end of a COBOL class definition is indicated by the END CLASS header.

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

© Copyright IBM Corp. 1991, 1998

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.

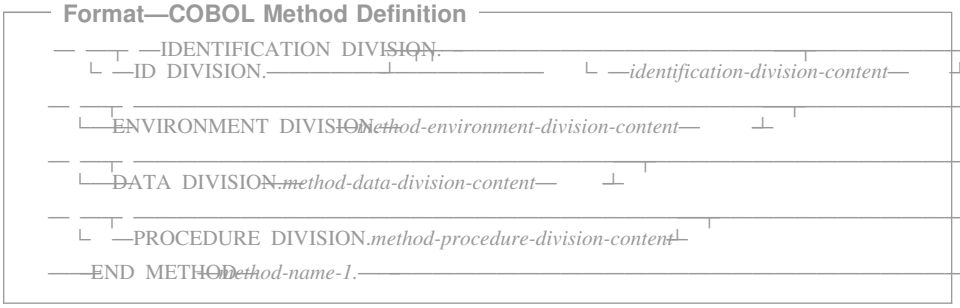
With the exception of the COPY and REPLACE statements and the END METHOD header, the statements, entries, paragraphs, and sections of a COBOL method definition are grouped into the following four divisions:

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, 1998

COBOL Method Definition

Part 3. Identification Division

Identification Division 74
PROGRAM-ID Paragraph 77
CLASS-ID Paragraph 79
METHOD-ID Paragraph 81
Optional Paragraphs 83

© Copyright IBM Corp. 1991, 1998

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.

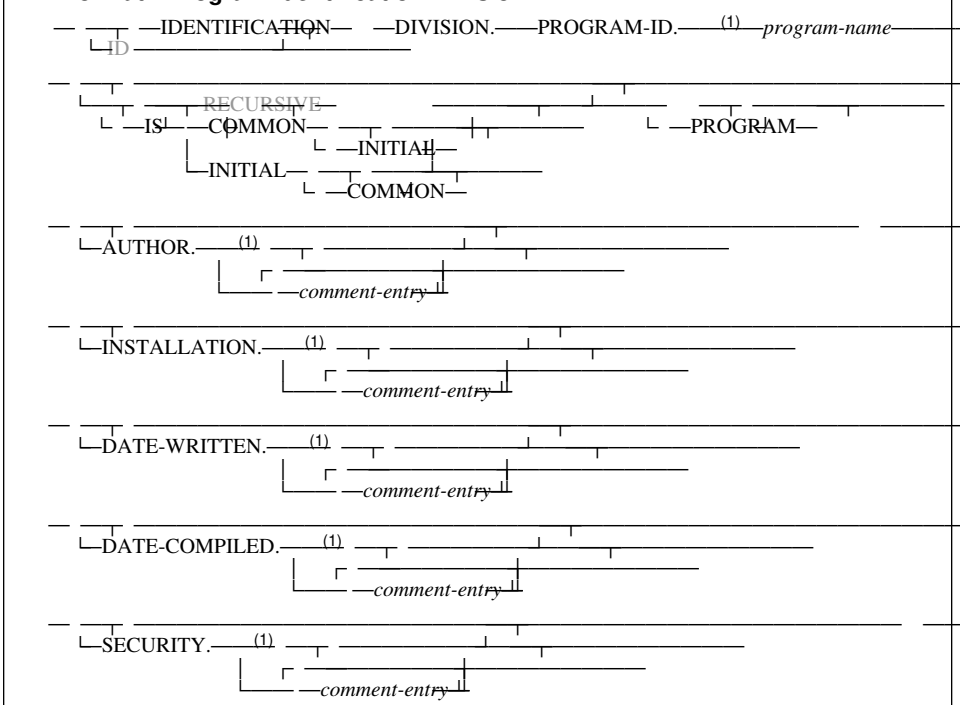
Method IDENTIFICATION DIVISION

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, 1998

Format—Program Identification Division

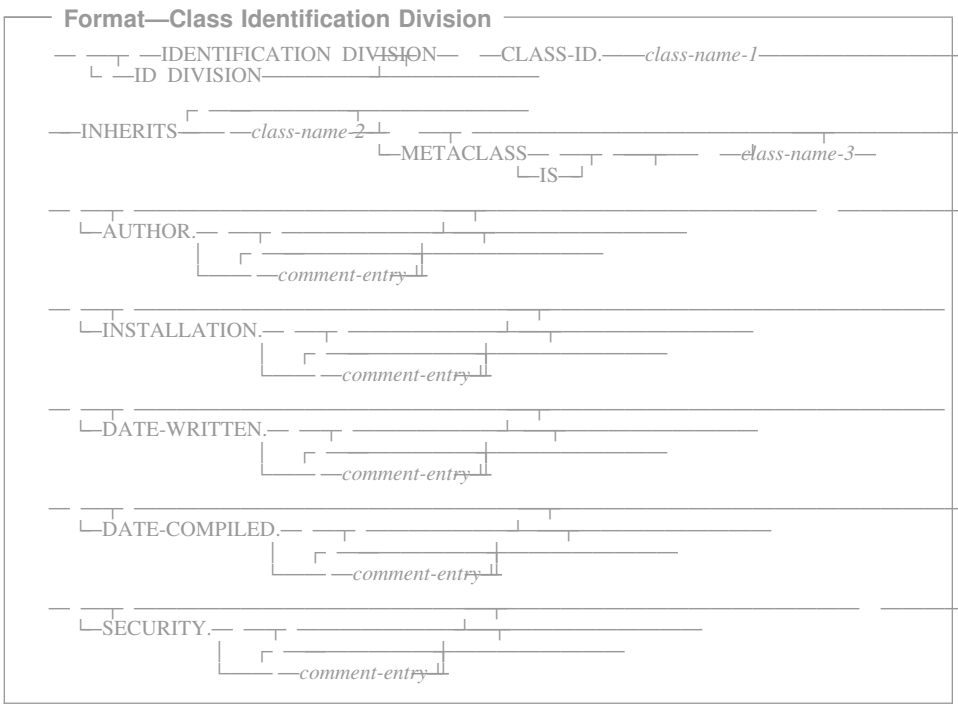


Note:

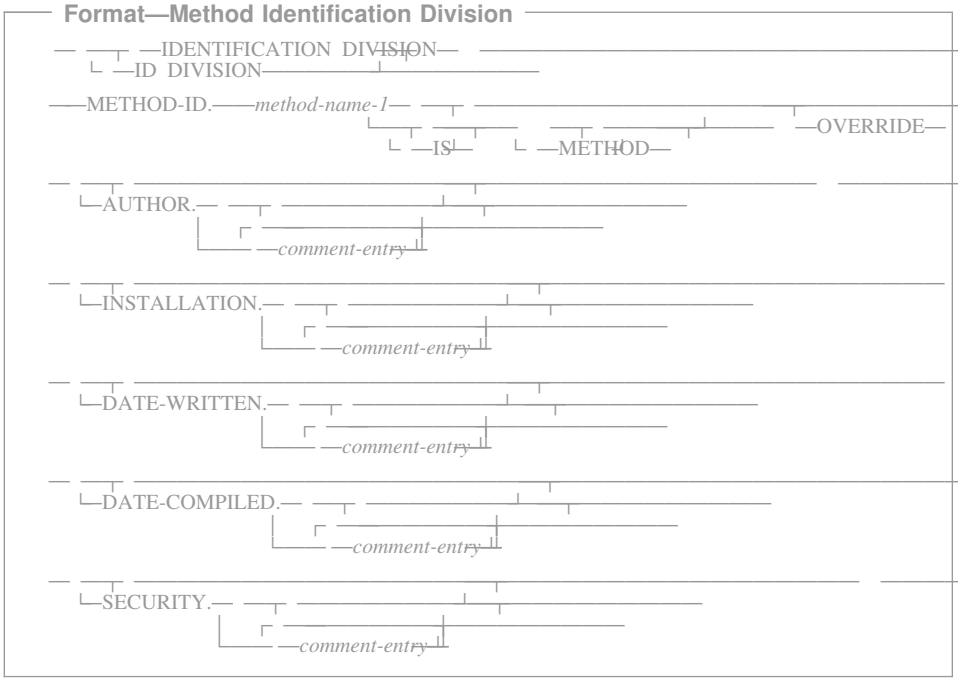
1 This separator period is optional as an IBM extension.

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

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	
	OS/390 and VM	**AIX, OS/2, and Windows**
PGMNAME (COMPAT)	The name can be up to 30 characters in length. Only the hyphen, digit, and alphabetic characters are allowed in the name. At least one character must alphabetic. The hyphen cannot be used as the first or last character. If program-name is a nonnumeric literal, (other than a figurative constant), it can include the extension characters $, #, and @ 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. If program-name is a nonnumeric literal, it can be up to 160 characters in length. It cannot be a figurative constant. Only the hyphen, digit, and alphabetic characters are allowed in the name. At least one character must alphabetic. The hyphen cannot be used as the first or last character.	
PGMNAME (LONGMIXED)	Program-name must be specified as a literal. It cannot be a figurative constant. The name can be up to 160 characters in length. Program-name can consist of any character in the range X'41' to X'FE'.	Program-name must be specified as a literal. It cannot be a figurative constant. The name can be up to 160 characters in length. Wherever alphabetic characters are allowed, you can use multi-byte characters.

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

RECURSIVE

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

You can specify the RECURSIVE clause only on the outermost program of a compilation 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 67.

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.

CLASS-ID Paragraph

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 3.

See “REPOSITORY Paragraph” on page 98 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 PICTURE clause currency symbols 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.

METHOD-ID Paragraph

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 PICTURE clause currency symbols 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 PICTURE clause currency symbols 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.  4/27/95.
```

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.

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

Host Under OS/390 and VM, you can include DBCS character strings as comment-entries in the Identification Division of your program. Multiple lines are allowed in a comment-entry containing DBCS strings.

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

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

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

DBCS strings are described under “Character-Strings” on page 3.

Part 4. Environment Division

Configuration Section 86
SOURCE-COMPUTER Paragraph 87
OBJECT-COMPUTER Paragraph 88
SPECIAL-NAMES Paragraph 89
ALPHABET Clause 92
SYMBOLIC CHARACTERS Clause 95
CLASS Clause 95
CURRENCY SIGN Clause 96
REPOSITORY Paragraph 98

Input-Output Section 100
FILE-CONTROL Paragraph 102
SELECT Clause 106
ASSIGN Clause 106
RESERVE Clause 110
ORGANIZATION Clause 111
PADDING CHARACTER Clause 114
RECORD DELIMITER Clause 114
ACCESS MODE Clause 115
RECORD KEY Clause 117
ALTERNATE RECORD KEY Clause 118
RELATIVE KEY Clause 119
PASSWORD Clause 120
LOCK MODE Clause (OS/2 VSAM Files Only) 120
FILE STATUS Clause 122
I-O-CONTROL Paragraph 124
RERUN Clause 125
SAME AREA Clause 127
SAME RECORD AREA Clause 127
SAME SORT AREA Clause 128
SAME SORT-MERGE AREA Clause 129
MULTIPLE FILE TAPE Clause 129
APPLY WRITE-ONLY Clause 129

© Copyright IBM Corp. 1991, 1998

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

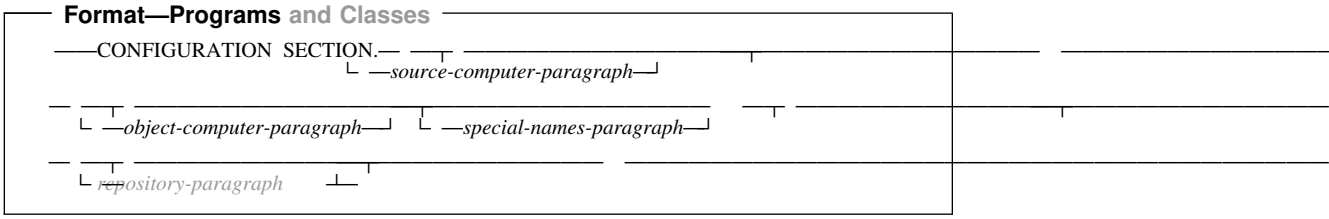
Specify the Configuration Section only in the Environment Division of the outermost program of a class definition.

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 currency sign value, and the currency symbol used in the PICTURE clause to represent the currency sign value

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

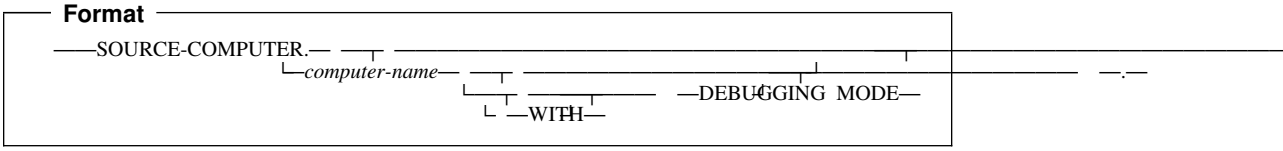
Relate object-oriented class names to the class names in the SOM interface repository

© Copyright IBM Corp. 1991, 1998

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-39

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.

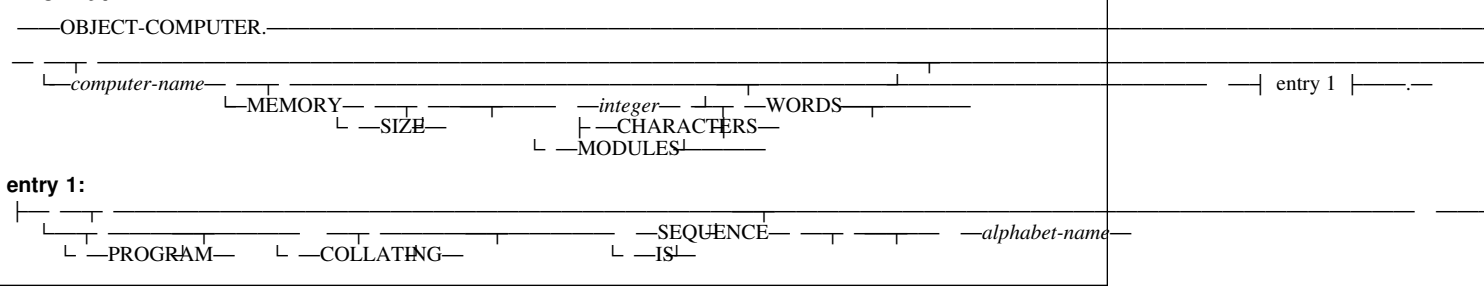
If a debugging line contains only spaces in Area A and in Area B, it is treated the same as a blank line.

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-39

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.

Host Under OS/390 and VM, the PROGRAM COLLATING SEQUENCE clause is not applied to the DBCS character set. Host

Workstation Under AIX, OS/2, and Windows, the PROGRAM COLLATING SEQUENCE clause is not allowed if the code page in effect is a DBCS or EUC code page. Workstation

When the PROGRAM COLLATING SEQUENCE clause is omitted:

Host Under OS/390 and VM, the EBCDIC collating sequence is used. (See Appendix B, “EBCDIC and ASCII Collating Sequences” on page 548.) Host

Workstation Under AIX, OS/2, 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. Workstation

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

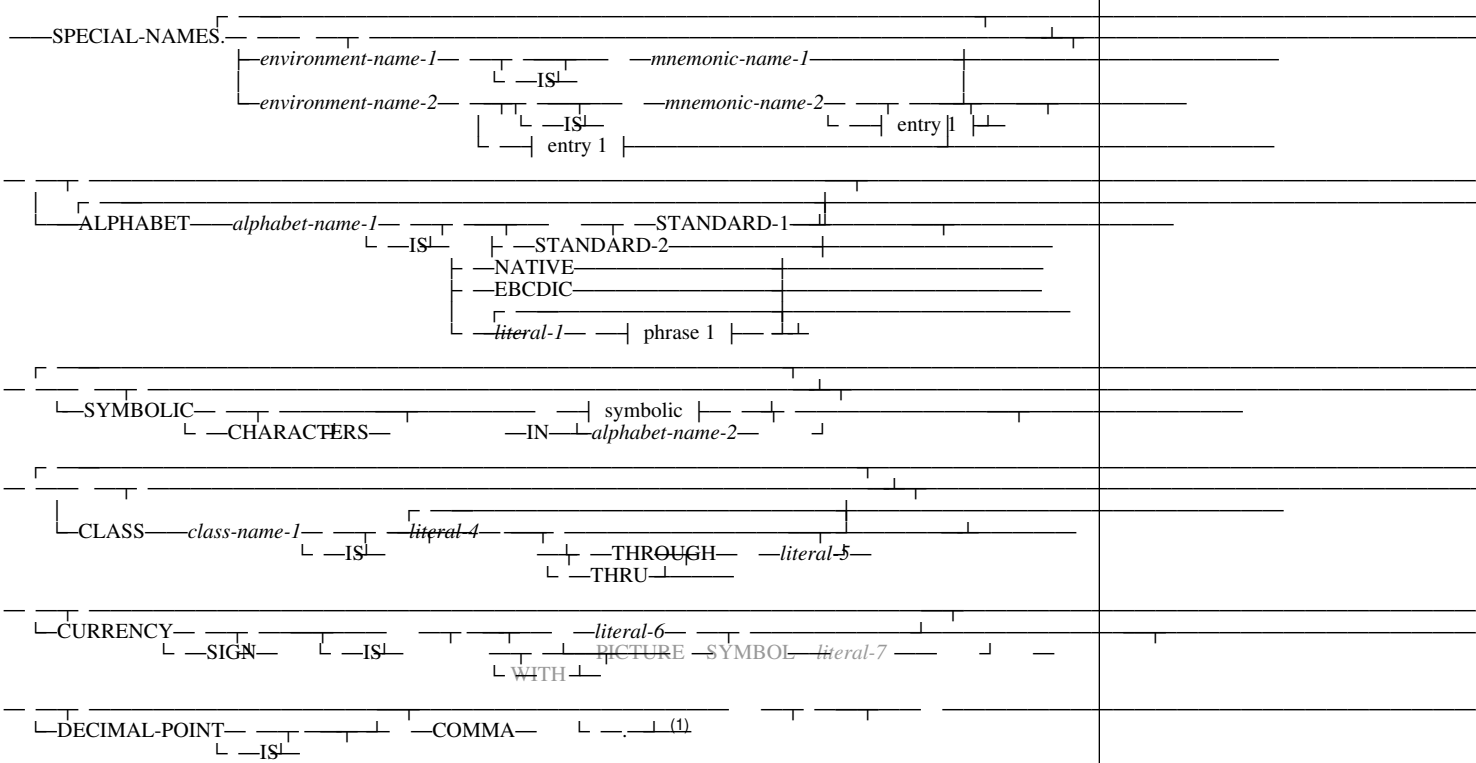
The SPECIAL-NAMES paragraph:

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 currency sign value, and the currency symbol used in the PICTURE clause to represent the currency sign value (multiple currency sign values and currency symbols may be specified)
Specifies that the functions of the comma and decimal point are to be interchanged in PICTURE clauses and numeric literals

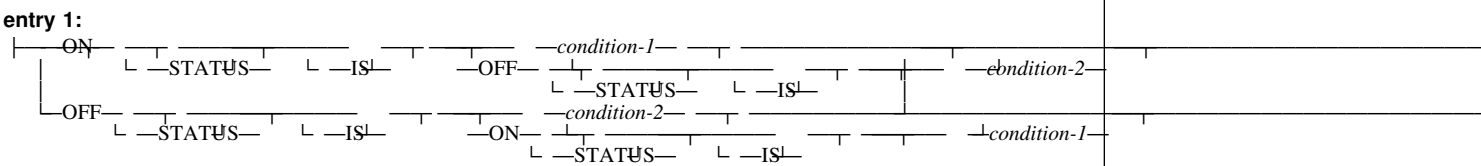
Note: The clauses in the SPECIAL-NAMES paragraph can appear in any order.

SPECIAL-NAMES Paragraph

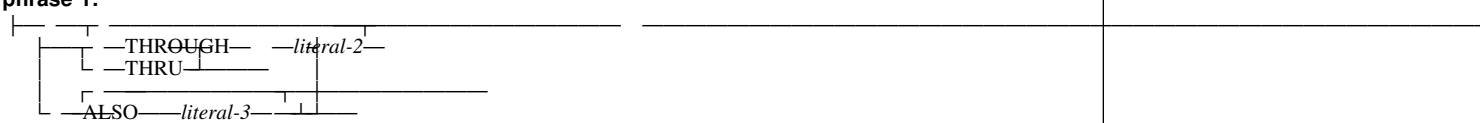
Format



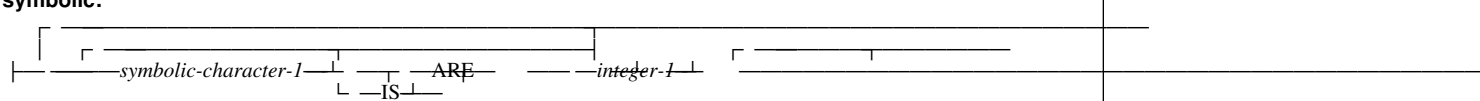
entry 1:



phrase 1:



symbolic:



Note:

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 SYSIPT	System logical input unit	ACCEPT
SYSOUT SYSLIST SYSLST	System logical output unit	DISPLAY
SYSPUNCH 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 254.)

ALPHABET Clause

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

Workstation Under AIX, OS/2, and Windows, you cannot specify the ALPHABET clause if the code page in effect is a DBCS or EUC code page. For details, see the *IBM COBOL Programming Guide* for your platform. Workstation

STANDARD-1

Host Under OS/390 and VM, specifies the ASCII character set. Host

Workstation 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. Workstation

STANDARD-2

Host Under OS/390 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. Host

Workstation 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. Workstation

NATIVE
Specifies the native character code set. If the alphabet-name clause is omitted:

Host Under OS/390 and VM, EBCDIC is assumed. Host

Workstation Under AIX, OS/2, and Windows, the alphabet-name is associated with the character set (ASCII or EUC) indicated by the locale in effect. Workstation

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 548, 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 159).

Literal-1, **literal-2**, and **literal-3** must not specify a symbolic-character figurative constant.

Floating-point literals cannot be used in a user-specified collating sequence.

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.

Host Under OS/390 and VM, the symbolic character can be a DBCS user-defined word. Host

Workstation Under AIX, OS/2, and Windows, you cannot use the SYMBOLIC CHARACTERS clause if the code page is DBCS or EUC. Workstation

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

Workstation Under AIX, OS/2, and Windows, you cannot specify the CLASS clause if the code page in effect is a DBCS or EUC code page. Workstation

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.

Host Under OS/390 and VM, the class-name in the CLASS clause can be a DBCS user-defined word. Host

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

CURRENCY SIGN Clause

the EBCDIC or ASCII collating series. Cannot be specified as floating-point literals 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

The CURRENCY SIGN clause affects numeric-edited data items whose PICTURE clause character-strings contain a *currency symbol*. A currency symbol represents a *currency sign value* that is:

Inserted in such data items, when they are used as receiving items

Removed from such data items, when they are used as sending items for a numeric or numeric-edited receiver

Typically, currency sign values identify the monetary units stored in a data item. For example: '$', 'EUR', 'FRF', 'F', 'HK$', 'HKD', or X'9F' (hexadecimal code point in some host-based code pages for € , the Euro currency sign; for more details on programming techniques for handling the Euro, see the *IBM COBOL Programming Guide* for your platform).

The CURRENCY SIGN clause specifies a currency sign value and the currency symbol used to represent that currency sign value in a PICTURE clause.

The SPECIAL-NAMES paragraph may contain multiple CURRENCY SIGN clauses. Each CURRENCY SIGN clause must specify a different currency symbol. Unlike all other PICTURE clause symbols, currency symbols are case-sensitive: for example, 'D' and 'd' specify different currency symbols.

CURRENCY SIGN IS literal-6
Literal-6 must be a nonnumeric literal. Literal-6 must not be a figurative constant, a DBCS literal, or a null-terminated literal.

If the PICTURE SYMBOL phrase is **not** specified, literal-6:

Specifies both a currency sign value and the currency symbol for this currency sign value.

Must be a single character.

Must not be any of the following:

- Digits 0 through 9
- Alphabetic characters A, B, C, D, E, G, N, P, R, S, V, X, Z, their lowercase equivalents, or the space
- Special characters + - , . * / ; ( ) " = '

Can be one of the following lowercase alphabetic characters: f, h, i, j, k, l, m, o, q, t, u, w, y

If the PICTURE SYMBOL phrase is specified, literal-6:

Specifies a currency sign value. Literal-7, in the PICTURE SYMBOL phrase, specifies the currency symbol for this currency sign value.

May consist of one or more characters.

Must not contain any of the following:

- Digits 0 through 9
- Special characters + - . ,

PICTURE SYMBOL literal-7

Specifies a currency symbol, which can be used in a PICTURE clause to represent the currency sign value specified by literal-6.

Literal-7 must be a nonnumeric literal consisting of a single character. Literal-7 must not be any of the following:

A figurative constant

Digits 0 through 9

Alphabetic characters A, B, C, D, E, G, N, P, R, S, V, X, Z, their lowercase equivalents, or the space

Special characters + - , . * / ; ( ) " = '

If the CURRENCY SIGN clause is specified, the CURRENCY and NOCURRENCY compiler options are ignored. If the CURRENCY SIGN clause is not specified and the NOCURRENCY compiler option is in effect, the dollar sign ($) is used as the default currency sign value and currency symbol. For more information about the CURRENCY and NOCURRENCY compiler options, see the *IBM COBOL Programming Guide* for your platform.

Some uses of the CURRENCY SIGN clause prevent use of the NUMVAL-C intrinsic function. For details, see “NUMVAL-C” on page 488.

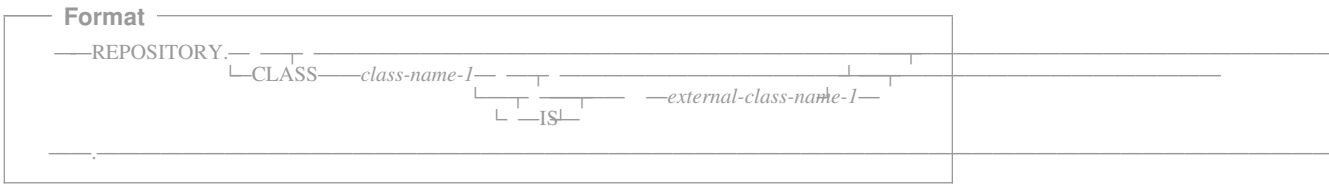
DECIMAL-POINT IS COMMA

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

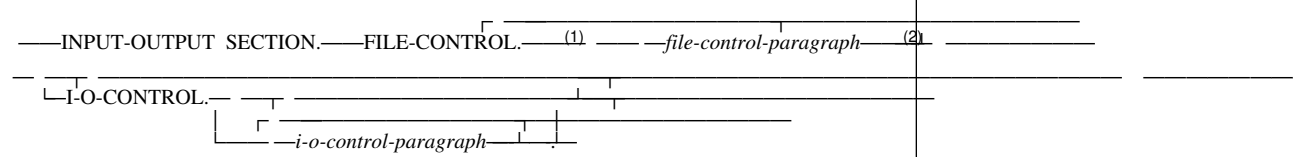
The exact contents of the Input-Output Section depend on the file organization and access methods used. See “ORGANIZATION Clause” on page 111 and “ACCESS MODE Clause” on page 115.

Program Input-Output Section
The same rules apply to program and method I-O Sections.

Class Input-Output Section
The Input-Output Section is **not** valid for class definitions.

Method Input-Output Section
The same rules apply to program and method I-O Sections.

Programs and Methods



Notes:
1 If there are no files defined in the program and the INPUT-OUTPUT SECTION is specified and no file-control-paragraph is specified, then the FILE-CONTROL paragraph-name is optional as an IBM extension.
2 If there are no files defined in the program and the FILE-CONTROL paragraph-name is specified, then the file-control-paragraph is optional as an IBM extension.

FILE-CONTROL
The key word FILE-CONTROL names the FILE-CONTROL paragraph. This key word 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 102.

I-O-CONTROL
The key word I-O-CONTROL names the I-O-CONTROL paragraph.

© Copyright IBM Corp. 1991, 1998

Input-Output Section

input-output-control-paragraph

Specifies information needed for efficient transmission of data between the external data set and the COBOL program. The series of entries must end with a separator period. See “I-O-CONTROL Paragraph” on page 124.

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
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 OS/390 and VM	File Systems		
		AIX	OS/2	Windows
Sequential	QSAM, VSAM	VSAM[1], STL	VSAM, Btrieve, STL	VSAM[2], Btrieve, STL
Relative	VSAM	VSAM[1], STL	VSAM, Btrieve, STL	VSAM[2], Btrieve, STL
Indexed	VSAM	VSAM[1], STL	VSAM, Btrieve, STL	VSAM[2], Btrieve, STL
Line Sequential	n/a	Native	Native	Native

Note:

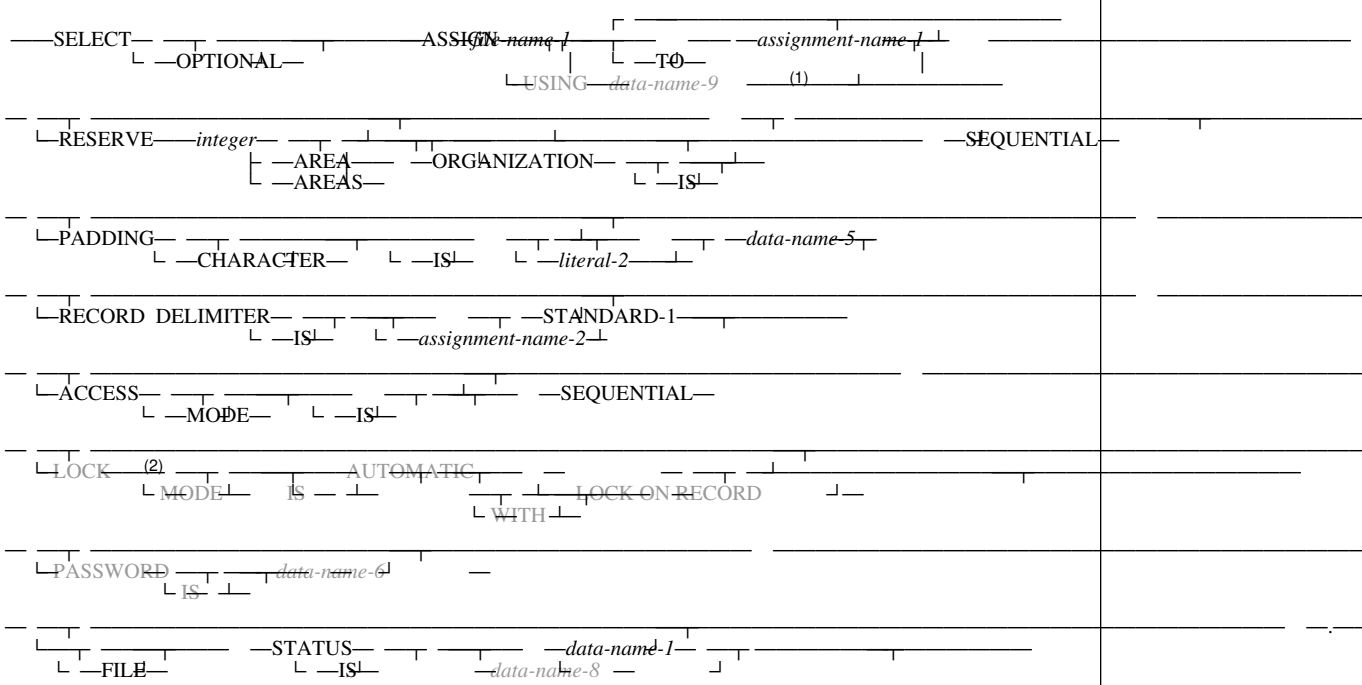
[1] On AIX, you can access the SFS file system through VSAM.

[2] On Windows, only remote file access is available.

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.

Host Under OS/390 and VM, there is one exception to the rule about order. For indexed files, the PASSWORD clause, if specified, must immediately follow the RECORD KEY or ALTERNATE RECORD KEY data-name with which it is associated. Host

Format 1—Sequential-File-Control-Entries



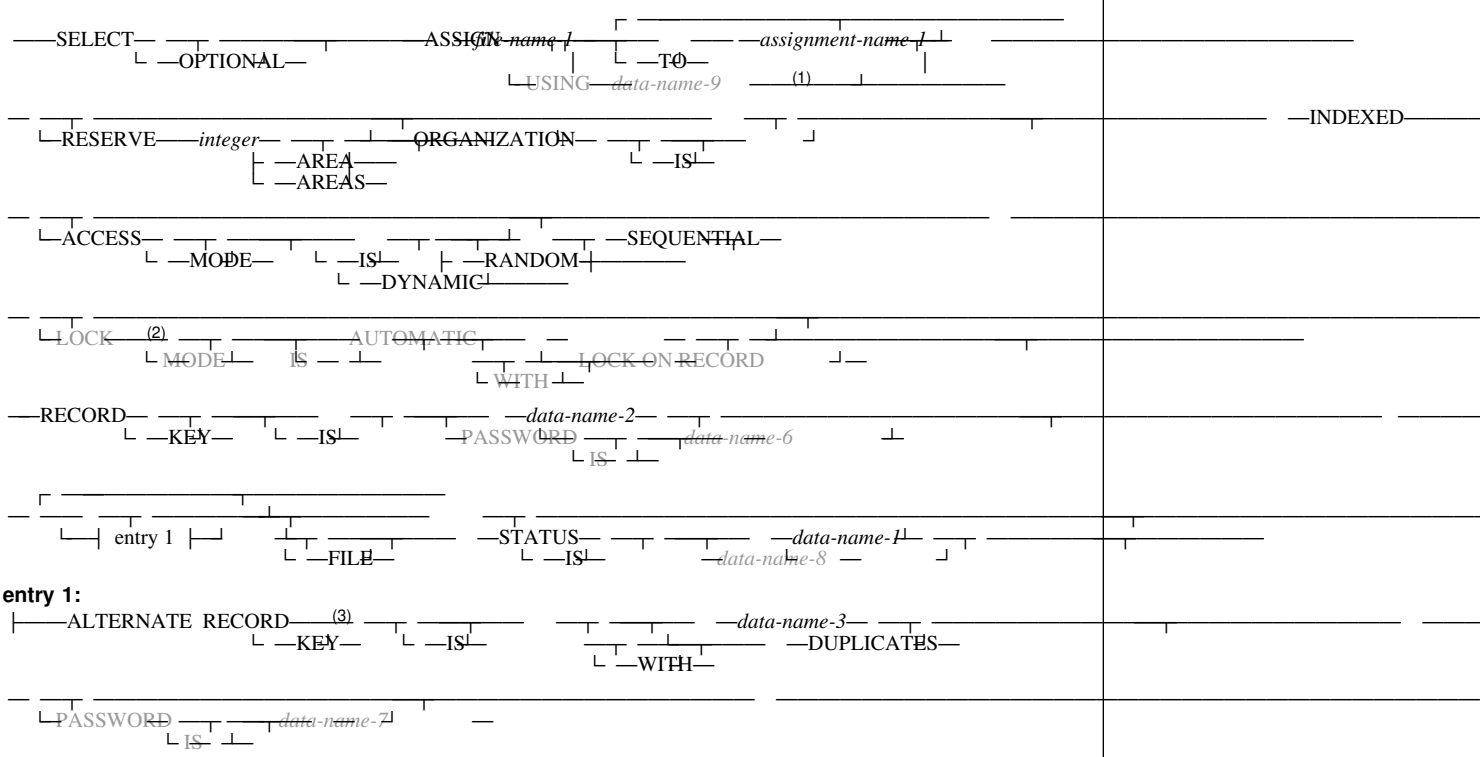
Notes:

1 The USING data-name phrase of the ASSIGN clause clause is only valid under AIX, OS/2, and Windows.

2 The LOCK clause is only meaningful for OS/2 VSAM files. It is treated as a comment on AIX, Windows, and files other than VSAM on OS/2. It is invalid on OS/390 and VM.

FILE-CONTROL Paragraph

Format 2—Indexed-File-Control-Entries



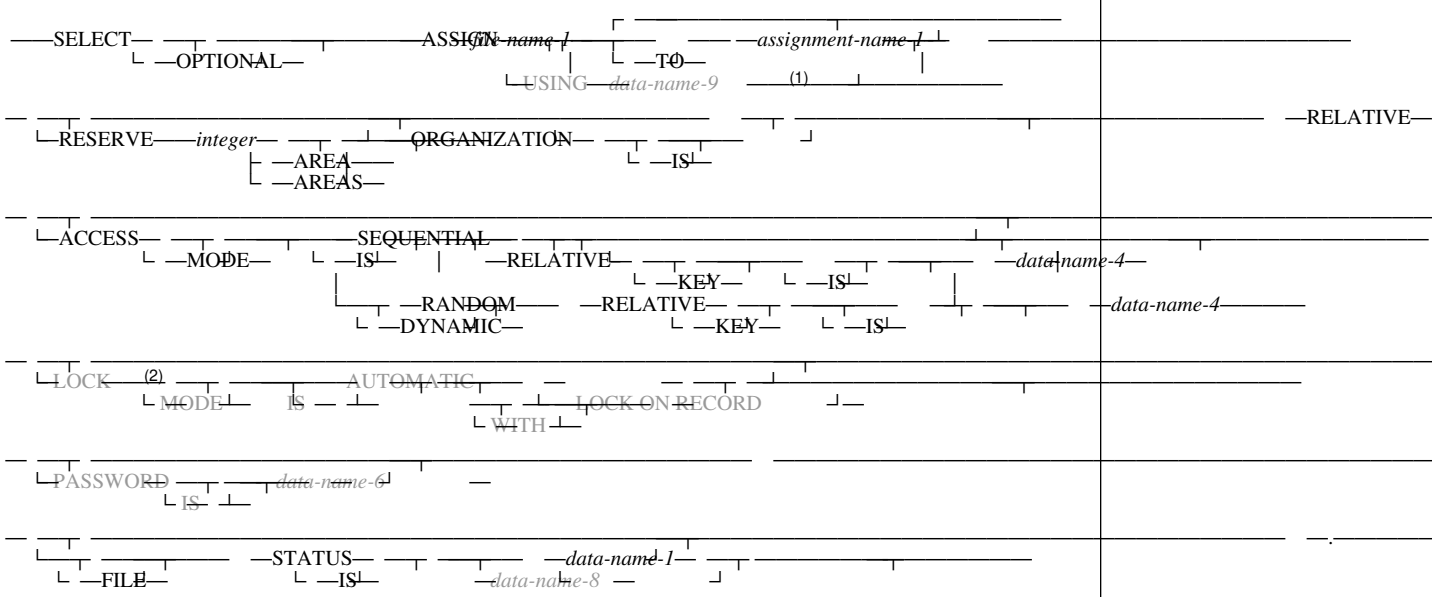
Notes:

1 The USING data-name phrase of the ASSIGN clause is only valid under AIX, OS/2, and Windows.

2 The LOCK clause is only meaningful for OS/2 VSAM files. It is treated as a comment on AIX, Windows, and files other than VSAM on OS/2. It is invalid on OS/390 and VM.

3 RECORD is optional as an IBM extension.

Format 3—Relative-File-Control-Entries

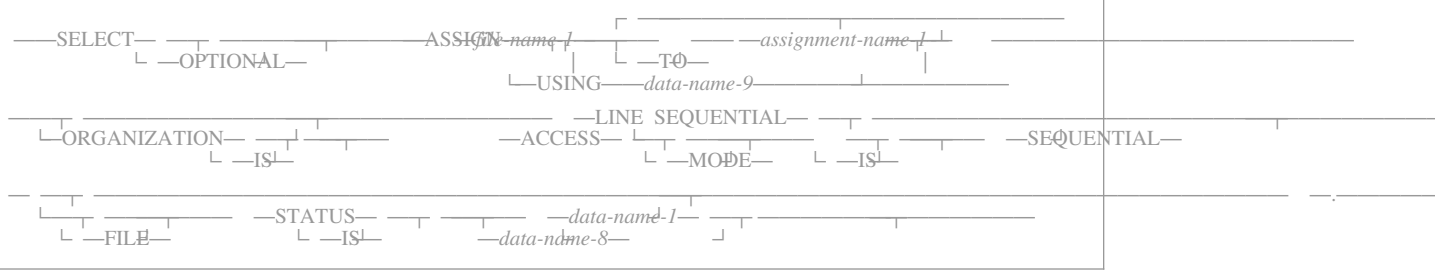


Notes:

1 The USING data-name phrase of the ASSIGN clause is only valid under AIX, OS/2, and Windows.

2 The LOCK clause is only meaningful for OS/2 VSAM files. It is treated as a comment on AIX, Windows, and files other than VSAM on OS/2. It is invalid on OS/390 and VM.

Format 4—Line Sequential I-O (Workstation Only)



ASSIGN Clause

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-1
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-1 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

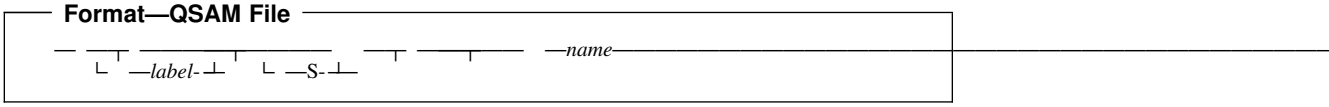
The ASSIGN clause associates the program's name for a file with the external name for the actual data file.

OS/390 and VM Syntax

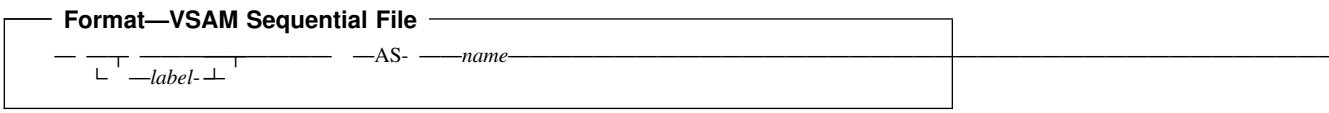
assignment-name-1
Can be specified as a user-defined word or a nonnumeric literal. Any assignment-name after the first is syntax checked, but it has no effect on the execution of the program.

Assignment-name-1 has the following formats:

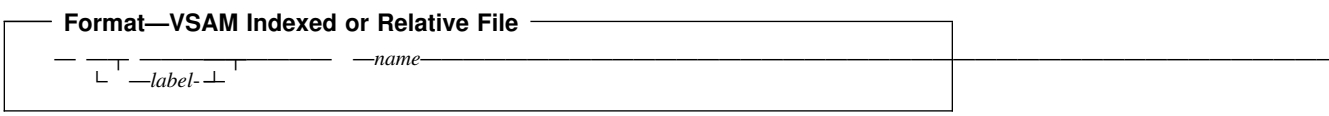
Format—QSAM File



Format—VSAM Sequential File



Format—VSAM Indexed or Relative File



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 OS/390 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 109 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.

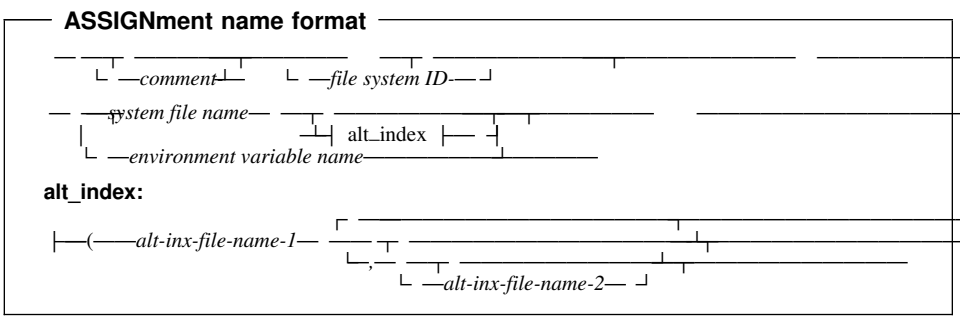
All characters specified within the literal delimiters are used without any mapping.

USING data-name-9

Must be defined as an alphanumeric data item, and must not be subordinate to the file description for file-name-1. The content is evaluated when OPENed to identify the assignment name. See "Assignment Name for Data-Names and Environment Variables" on page 109 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:



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 the entire character string (along with any character strings to the left of it) is treated as a comment. If you include com-

ments (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— 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 OS/390 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)

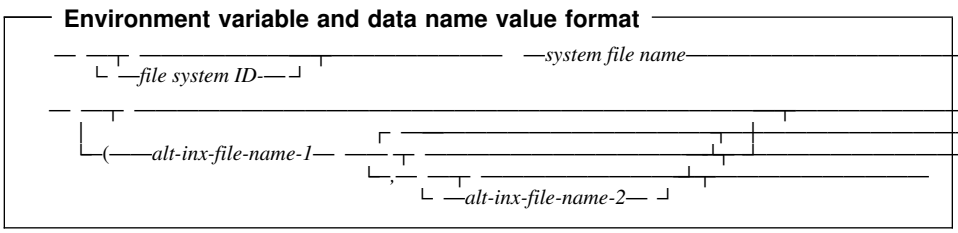
In the above example, the compiler will assign a default file name for the second alternate index file.

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:

RESERVE Clause



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 109.

RESERVE Clause

Workstation Under AIX, OS/2, and Windows, the RESERVE clause is not supported for line sequential files. It is treated as a comment for sequential, relative and indexed files. Workstation

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 OS/390. If none is specified, the system default is taken.

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 116.

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.

ORGANIZATION IS LINE SEQUENTIAL (Format 4) (Workstation Only)
Under AIX, OS/2, and Windows, 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. 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.

Line Sequential Organization (Workstation Only)

In a line sequential file, each record contains a sequence of characters ending with a record terminator. The terminator is not counted in the length of the record. When records are written to the file, trailing blanks are removed.

When reading the record, characters are read one at a time into the record area until:

- The first record terminator is encountered. The record terminator is discarded and the remainder of the record is filled with spaces.
- The entire record area is filled with characters. If the first unread character is the record terminator, it is discarded. Otherwise, the first unread character becomes the first character read by the next READ statement.

Records written to line sequential files must consist of USAGE...DISPLAY and/or DISPLAY-1 data items. If external decimal data is defined with a non-separate sign,

the sign must be in the preferred sign representation (for example, X'39' for +9 or X'79' for -9).

For line sequential files, the native byte stream file support of the platform is used. Line sequential files should contain only printable characters and the record terminator.

The following are not supported for line sequential files:

APPLY WRITE ONLY clause
BLOCK CONTAINS clause
CODE-SET clause
DATA RECORDS clause
FILE STATUS value 39 (fixed file attribute conflict)
LABEL RECORDS clause
LINAGE clause
OPEN I-O option
PADDING CHARACTER clause
RECORD CONTAINS 0 clause
RECORD CONTAINS clause (format 3)
RECORD DELIMITER clause
RECORDING MODE clause
RERUN clause
RESERVE clause
REVERSED phrase of OPEN statement
REWRITE statement
VALUE OF clause of file description entry
WRITE...AT END-OF-PAGE
WRITE...BEFORE ADVANCING

For more details on line sequential files, see “Line Sequential Organization (Workstation Only)” on page 112.

Language Elements Treated as Comments (Workstation Only)

Under AIX, OS/2, and Windows for other files (sequential, relative, and indexed), the following language elements are treated as comments:

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
PASSWORD clause
RECORD CONTAINS 0 clause
RECORD DELIMITER clause
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

Workstation Under AIX, OS/2, and Windows, the PADDING CHARACTER clause is not supported for line sequential files. It is treated as a comment for sequential, relative and indexed files. Workstation

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 1-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

Workstation Under AIX, OS/2, and Windows the RECORD DELIMITER clause is not supported for line sequential files. It is treated as a comment for sequential, relative and indexed files. Workstation

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.

For sequentially accessed relative files, the ACCESS MODE clause does not have to 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.

Format 4—Line Sequential (Workstation Only)
Workstation Records in the file are accessed in the sequence established when the file is created or extended. Format 4 supports only sequential access. Workstation

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.

ACCESS MODE Clause

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 102.

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.

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.

Line Sequential Files
Same as for sequential files (described above).

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 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.

As an IBM extension, data-name-2 can be numeric, numeric-edited, alphanumeric-edited, alphabetic, floating-point (both external and internal), or a DBCS data item. The key is treated as an alphanumeric item for the input and output statements for the file named in the SELECT clause. When you specify data-name-2 as a DBCS 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.

As an IBM extension, if the indexed file contains variable-length records, data-name-2 need not be contained within the first “x” character positions of the

ALTERNATE RECORD KEY Clause

record, where “x” equals the minimum record size specified for the file. That is, data-name-2 can be beyond the first “x” character positions of the record, but this is not recommended.

Data-name-2 cannot be a windowed date field.

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.

The requirement for identical data description entries is not enforced, but the key must 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.

As an IBM extension, data-name-3 can be a numeric, numeric-edited, alphanumeric-edited, alphabetic, floating-point (both external and internal), or DBCS data item. The key is treated as an alphanumeric item for the input and output statements for the file named in the SELECT clause.

Data-name-3 must not reference a group item that contains a variable occurrence data item. Data-name-3 can be qualified.

As an IBM extension, if the indexed file contains variable-length records, data-name-3 need not be contained within the first “x” character positions of the record, where “x” equals the minimum record size specified for the file. That is, data-name-3 can be beyond the first “x” character positions of the record, but this is not recommended.

Data-name-3 cannot be a windowed date field.

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 left-most 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 same relative location within the associated record, the same number of alternate record keys, and the same DUPLICATES phrase.

The requirement for identical data description entries is not enforced, but the key must 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 cannot be a windowed date field.

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.

LOCK MODE Clause

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.

PASSWORD Clause

Workstation Under AIX, OS/2, and Windows the PASSWORD clause is treated as a comment. Workstation

The PASSWORD clause controls access to files.

data-name-6
data-name-7
Password data items. Each must be defined in the Working-Storage Section (of the Data Division) as an alphanumeric item. The first 8 characters are used as the password; a shorter field is padded with blanks to 8 characters. Each password data item must be equivalent to one that is externally defined.

When the PASSWORD clause is specified, at object time the PASSWORD data item 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 (OS/2 VSAM Files Only)

Host On OS/390 and VM, the LOCK MODE IS AUTOMATIC clause is invalid. Host

Workstation The LOCK MODE IS AUTOMATIC clause is only supported by the VSAM file system running on OS/2.

The LOCK MODE IS AUTOMATIC clause is treated as a comment on:

AIX
Windows
OS/2 (with the exception of VSAM)

For OS/2 VSAM files, record locking is not supported for files that reside on an OS/2 LAN server. Files residing on OS/2 LAN servers are opened shared read or exclusive write. ◀ Workstation

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 and is supported.

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 phrase is for documentation purposes only.

Other Statements Affecting Record Locking

Table 10 lists the statements that can affect record locking.

Table 10 (Page 1 of 2). 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. 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. 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. 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.

Table 10 (Page 2 of 2). Statements Affecting Record Locking

Statement	Comments
REWRITE	You cannot specify the REWRITE statement for a record that another file connector has LOCKed (the file is exclusive). 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: **Sequential files:** the records are in an unspecified order. **Relative files:** the relative key values returned are ascending but not necessarily consecutive. **Indexed files:** the order of the alternate keys allowing for duplicates is unspecified. 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 270.)

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.

Host Under OS/390 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 OS/390 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.

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*. Host

Workstation 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 System

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 struc-

I-O-CONTROL Paragraph

ture (defined by VSAM). If the size of the reply message structure (m) is shorter than 'n', only the first 'm' bytes contain useful information.

Note: This also applies to SFS files accessed through VSAM on AIX.

For information on VSAM file handling on the workstation, 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*

For Windows: *SMARTdata UTILITIES User's Guide for Windows* Workstation

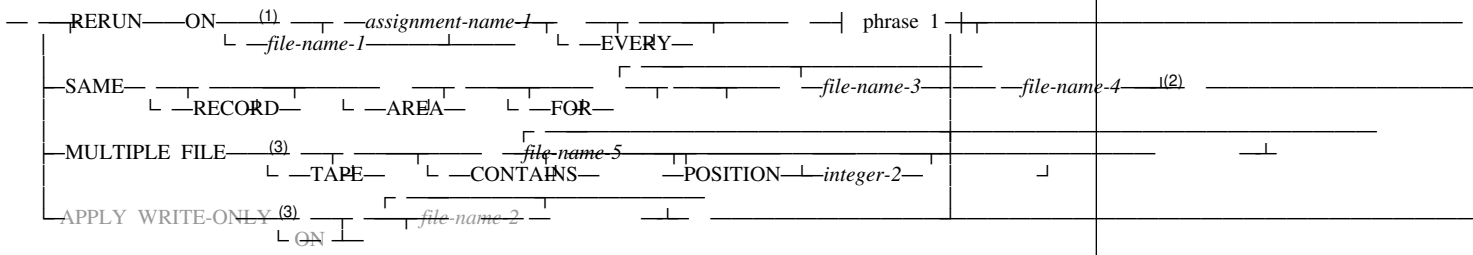
I-O-CONTROL Paragraph

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.

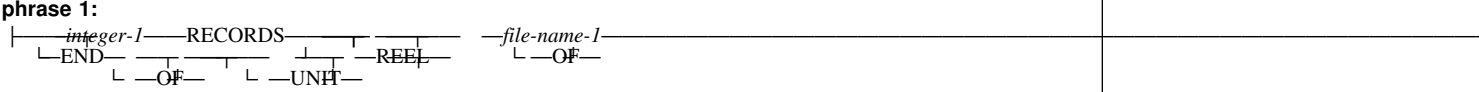
The key word I-O-CONTROL can appear only once, at the beginning of the paragraph. The word I-O-CONTROL must begin in Area A, and must be followed by a separator period.

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:



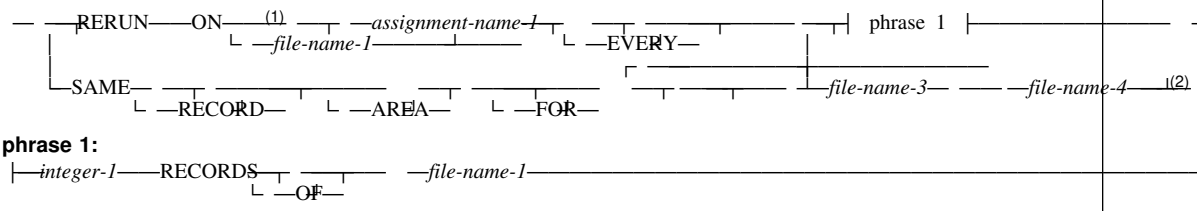
Notes:

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 OS/390 VSAM files and are treated as comments on AIX, OS/2, and Windows.

Relative and Indexed I-O-Control Entries



phrase 1:

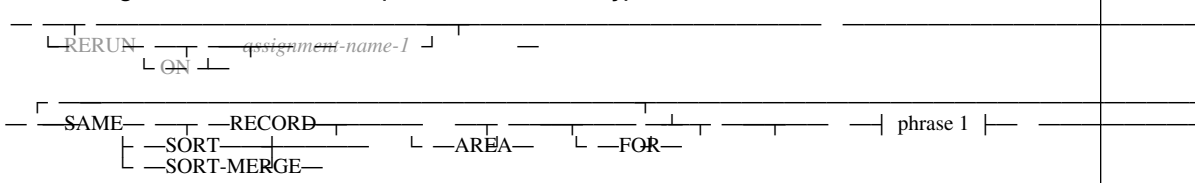
Notes:

1 ON is optional as an IBM extension.

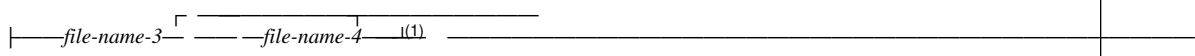
2 File-name-4 is optional as an IBM extension.

Line Sequential I-O-Control Entries (Workstation Only)

Sort Merge I-O-Control Entries (OS/390 and VM Only)



phrase 1:



Note:

1 File-name-4 is optional as an IBM extension.

RERUN Clause

Workstation 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. Workstation

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 OS/390 & VM Programming Guide*.

Do not use the RERUN clause:

On files with the EXTERNAL attribute
In programs with the RECURSIVE attribute

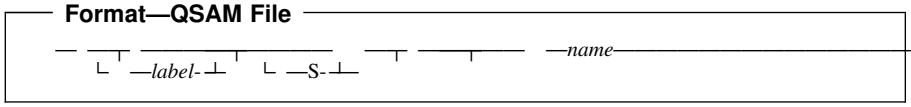
RERUN Clause

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 OS/390 and VM" on page 565.

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.

SORT/MERGE Considerations:

When the RERUN clause is specified in the I-O-CONTROL paragraph, checkpoint records are written at logical intervals determined by the sort/merge program during execution of each SORT or MERGE statement in the program. When it is omitted, checkpoint records are not written.

There can be only **one** SORT/MERGE I-O-CONTROL paragraph in a program, and it cannot be specified in contained programs. It will have a global effect on all 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.

Note: This clause is not supported. If you code it in your program, it will be treated as a comment.

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

Workstation Under AIX, OS/2, and Windows, the SAME AREA clause is treated as a comment. Workstation

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.
For OS/390 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.

SAME SORT 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

Workstation Under AIX, OS/2, and Windows, the SAME SORT AREA clause is treated as a comment. Workstation

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

Workstation Under AIX, OS/2, and Windows, the SAME SORT-MERGE AREA clause is treated as a comment. Workstation

The SAME SORT-MERGE AREA clause is equivalent to the SAME SORT AREA clause.

MULTIPLE FILE TAPE Clause

Workstation Under AIX, OS/2, and Windows, all files are treated as a single volume file. Any multiple volume files specified are treated as comments. Workstation

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

Workstation Under AIX, OS/2, and Windows, the APPLY WRITE-ONLY clause is not supported for line sequential files. It is treated as a comment for sequential, relative and indexed files. Workstation

The APPLY WRITE-ONLY clause optimizes buffer and device space allocation for files that have standard sequential organization, have variable-length records, and are blocked. If you specify this phrase, the buffer is truncated only when the space available in the buffer is smaller than the size of the next record. Otherwise, the buffer is truncated when the space remaining in the buffer is smaller than the maximum record size for the file.

APPLY WRITE-ONLY is effective only for QSAM files.

file-name-2
Each file must have standard sequential organization.

APPLY WRITE-ONLY clauses must agree among corresponding external file description entries. For an alternate method of achieving the APPLY WRITE-ONLY results, see the description of the AWO compiler option in the *IBM COBOL Programming Guide* for your platform.

APPLY WRITE-ONLY Clause

Part 5. Data Division

Data Division Overview 132
File Section 133
Working-Storage Section 133
Local-Storage Section 135
Linkage Section 135
Data Types 136
Data Relationships 137

Data Division—File Description Entries 144
File Section 147
EXTERNAL Clause 148
GLOBAL Clause 149
BLOCK CONTAINS Clause 149
RECORD Clause 151
LABEL RECORDS Clause 154
VALUE OF Clause 155
DATA RECORDS Clause 155
LINAGE Clause 155
RECORDING MODE Clause 157
CODE-SET Clause 159

Data Division—Data Description Entry 161
Format 1 161
Format 2 162
Format 3 162
Level-Numbers 162
BLANK WHEN ZERO Clause 164
DATE FORMAT Clause 164
EXTERNAL Clause 170
GLOBAL Clause 170
JUSTIFIED Clause 171
OCCURS Clause 172
PICTURE Clause 178
REDEFINES Clause 195
RENAMES Clause 198
SIGN Clause 200
SYNCHRONIZED Clause 202
USAGE Clause 209
VALUE Clause 217

© Copyright IBM Corp. 1991, 1998

Data Division Overview

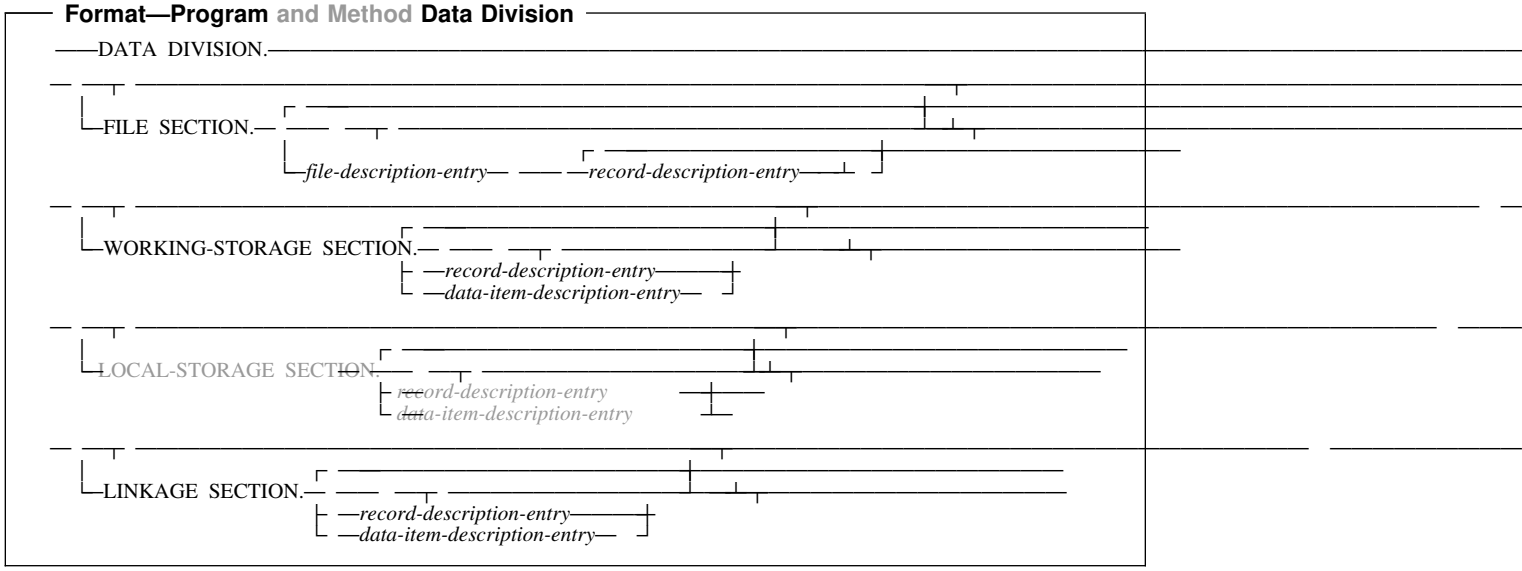
This overview describes the structure of the Data Division for programs, classes, and methods. Each section in the Data Division has a specific logical function within a COBOL source program or method and can be omitted when that logical function is not needed. If included, the sections must be written 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.

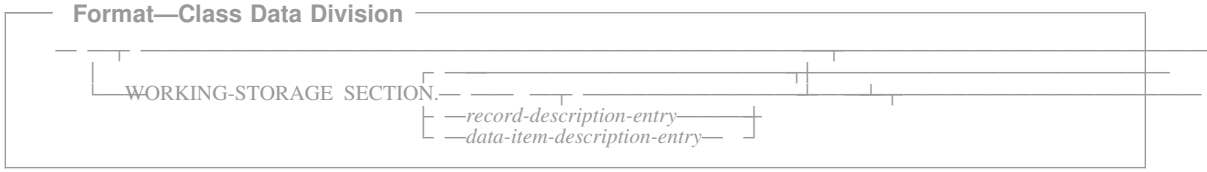
Class Data Division
The Class Data Division section contains data description entries for object-instance data. The Class Data Division contains only the Working-Storage Section.

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



© Copyright IBM Corp. 1991, 1998

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 144.

record-description-entry
A set of data description entries (described in "Data Division—Data Description Entry" on page 161) 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.

Note: A method File Section can define EXTERNAL files only. A single run-unit level file connector is shared by all programs and methods containing a declaration of a 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.

Method Working-Storage
A single copy of the Working-Storage for a method is statically allocated and persists in a last-used state for the duration of the run-unit. The same single copy is used whenever the method is invoked, regardless of which object the method is invoked upon.

If a VALUE clause is specified on a method Working-Storage data item, the data 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.
 1 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 161 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 161 for description.

Note: The data description entries for a class differ from a program and method in that:

You cannot specify the EXTERNAL attribute in a data description entry.
The GLOBAL attribute has no effect.
You can only specify the VALUE clause on condition names.

Local-Storage Section

The Local-Storage Section defines storage that is allocated and freed on a per-invocation basis. On each invocation, data items defined in the Local-Storage Section are reallocated and initialized to the value assigned in their VALUE clauses. Data items defined in the Local-Storage Section cannot specify the EXTERNAL clause.

The Local-Storage Section must begin with the header LOCAL-STORAGE SECTION followed by a separator period.

You can specify the Local-Storage Section in recursive programs, in non-recursive programs, and in methods.

Note: Method Local-Storage content is the same as a program Local-Storage content except that the GLOBAL attribute has no effect (since methods cannot be nested).

A separate copy of the data defined in a method Local-Storage section is created each time the method is invoked. The storage allocated for the data is freed when the 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 133 for description.

Data Types

data-item-description-entry
See “Working-Storage Section” on page 133 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.

As an IBM extension, you can specify the GLOBAL clause in the Linkage Section. (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 147.) 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 162 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.

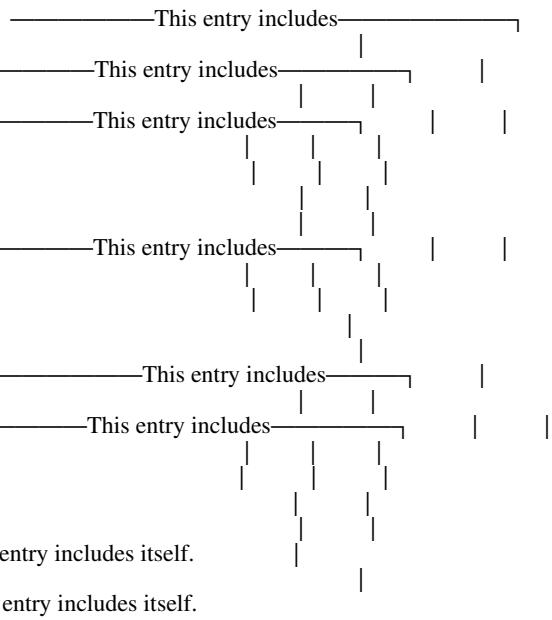
All elementary or group items immediately subordinate to one group item must be assigned identical level-numbers higher than the level-number of this group item.

Figure 4 on page 139 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 COBOL record description entry written as follows

```
1   RECORD—ENTRY.
  5   GROUP—1.
    1   SUBGROUP—1.
      15  ELEM—1  PIC...  .
      15  ELEM—2  PIC...  .
    1   SUBGROUP—2.
      15  ELEM—3  PIC...  .
      15  ELEM—4  PIC...  .
  5   GROUP—2.
      15  SUBGROUP—3.
        25 ELEM—5 PIC...  .
        25 ELEM—6 PIC...  .
      15  SUBGROUP—4 PIC...  .
  5   ELEM—7 PIC...  .
```

is subdivided as indicated below:



The storage arrangement of the record description entry is illustrated below:

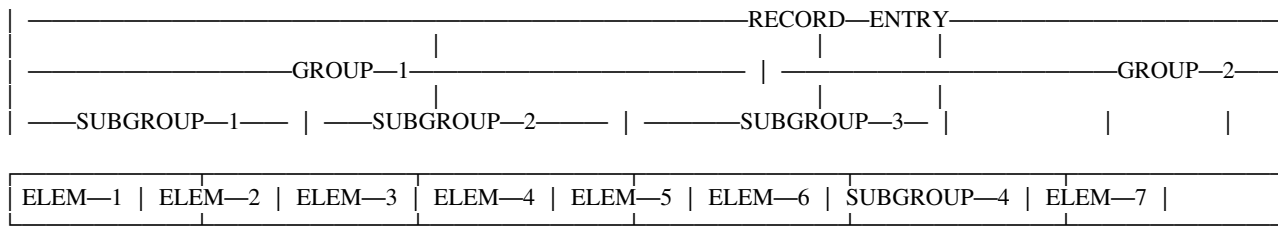


Figure 4. Levels in a Record Description

Data Relationships

IBM COBOL accepts nonstandard level-numbers that are not identical to others at the same level. For example, the following two record description entries are equivalent:

```
1    EMPLOYEE-RECORD.
       5   EMPLOYEE-NAME.
           1   FIRST-NAME  PICTURE   X(1 ).
           1   LAST-NAME   PICTURE   X(1 ).
       5   EMPLOYEE-ADDRESS.
           1   STREET      PICTURE   X(1 ).
           1   CITY        PICTURE   X(1 ).
1    EMPLOYEE-RECORD.
       5   EMPLOYEE-NAME.
           1   FIRST-NAME  PICTURE   X(1 ).
           1   LAST-NAME   PICTURE   X(1 ).
       4   EMPLOYEE-ADDRESS.
           8   STREET      PICTURE   X(1 ).
           8   CITY        PICTURE   X(1 ).
```

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 198.)

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 217.)

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 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
		Internal Floating-point
		External Floating-point
	Alphanumeric	Numeric-Edited
		Alphanumeric-Edited
		Alphanumeric
		DBCS
Group	Alphanumeric	Alphabetic
		Numeric
		Internal Floating-point
		External Floating-point
		Numeric-Edited
		Alphanumeric-Edited
		Alphanumeric
		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 353).

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.

Internal Floating-point
A decimal point is assumed immediately to the left of the field. The data is aligned then on the leftmost digit position following the decimal point, with the exponent adjusted accordingly.

External Floating-point
The data is aligned on the leftmost digit position; the exponent is adjusted 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 171.

Workstation 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. Workstation

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.

For internal floating-point items, the size of the item in storage is determined by its USAGE clause. USAGE COMPUTATIONAL-1 reserves 4 bytes of storage for the item; 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 209.

The TRUNC compiler option can affect the value of a binary numeric item. For information 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 209.) 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

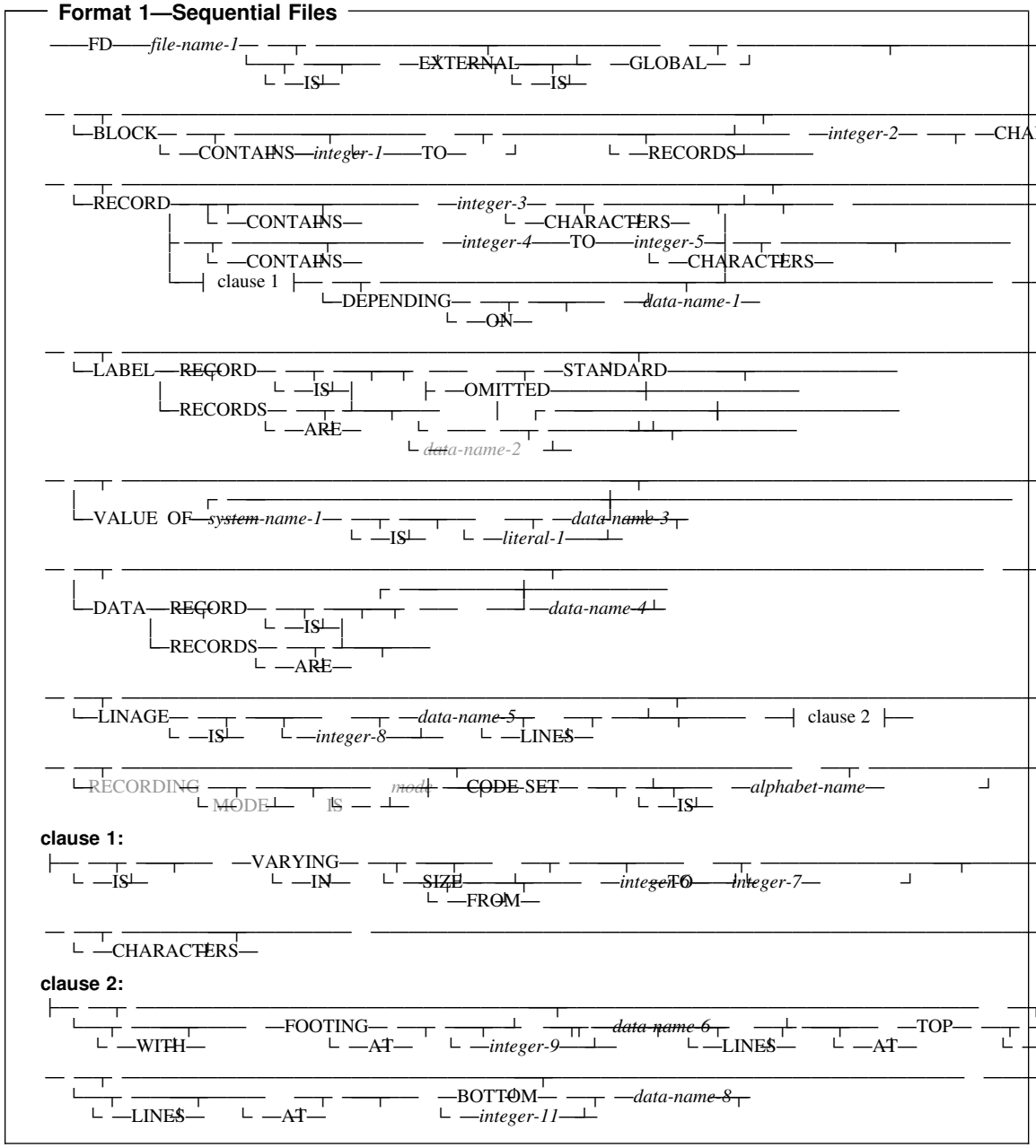
Editing signs are associated with numeric-edited items; editing signs are PICTURE symbols that identify the sign of the item in edited output.

Data Division—File Description Entries

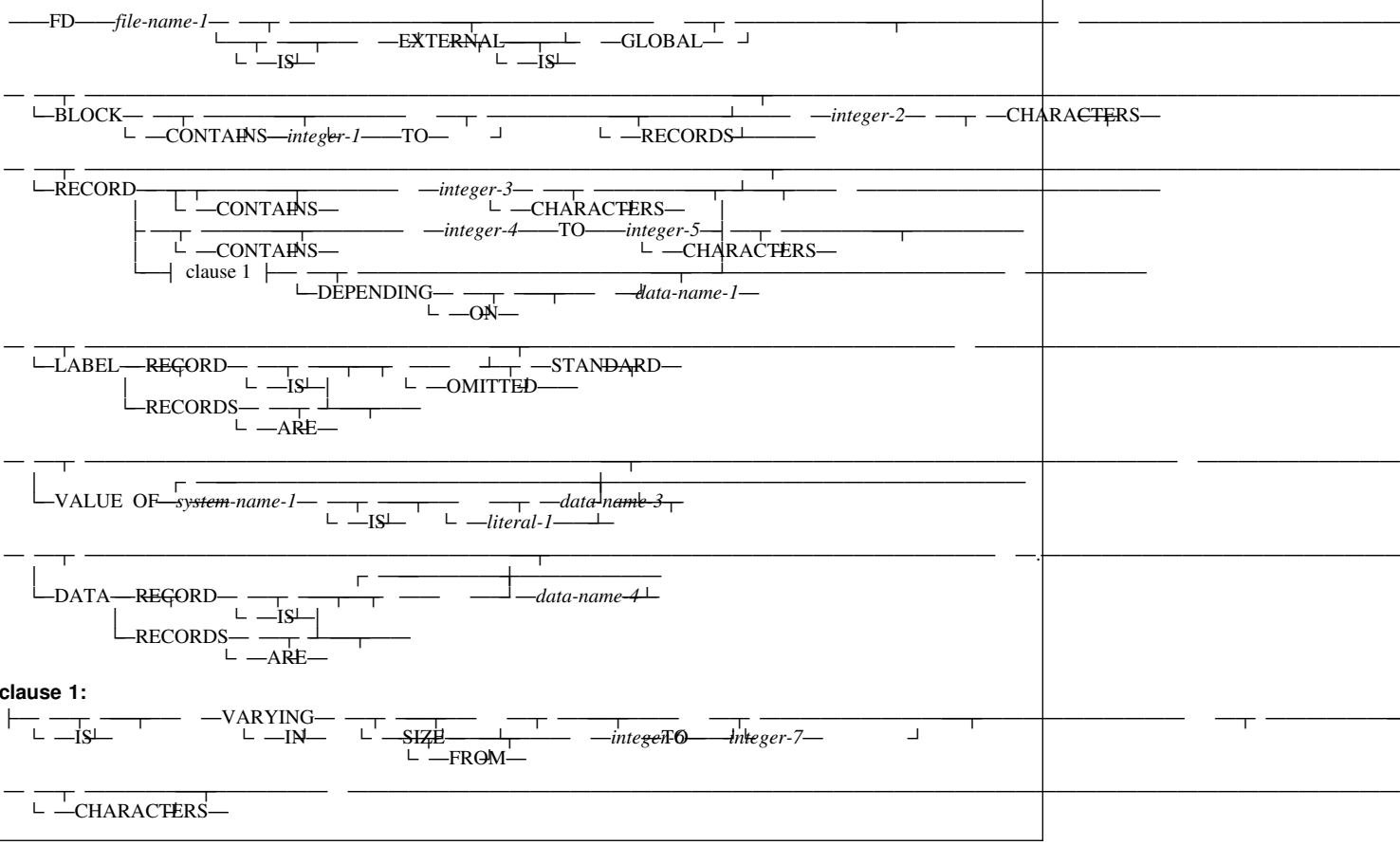
In a COBOL program, the **File Description (FD) Entry** (or **Sort File Description (SD) Entry** for sort/merge files) represents the highest level of organization in the File Section. The order in which the optional clauses follow the FD or SD entry is not important.

© Copyright IBM Corp. 1991, 1998

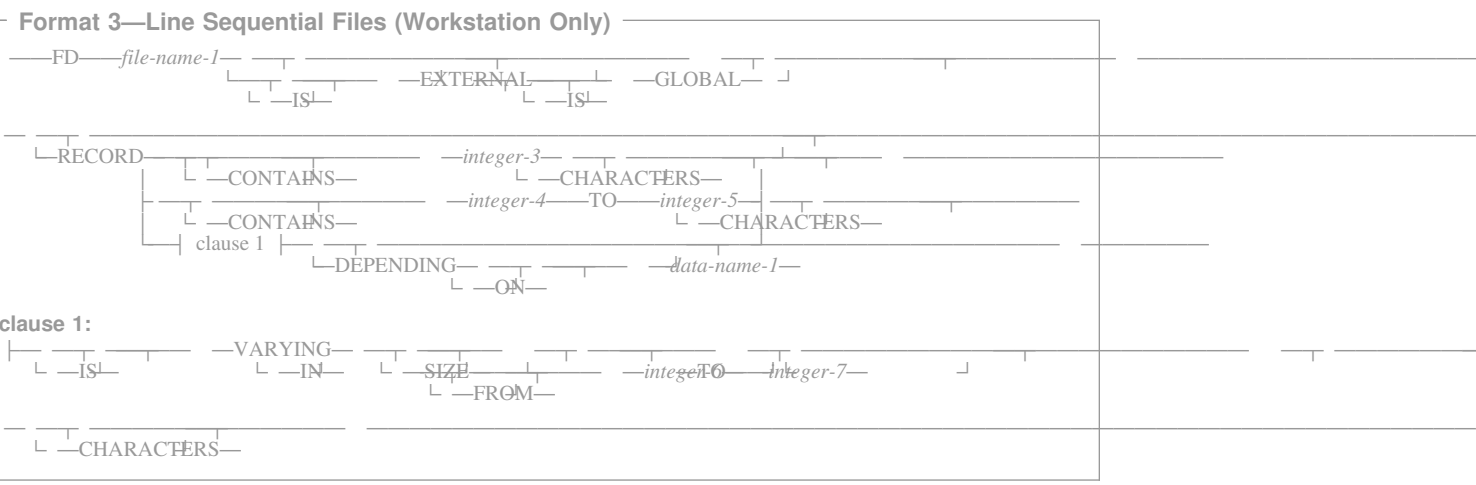
Format 1—Sequential Files



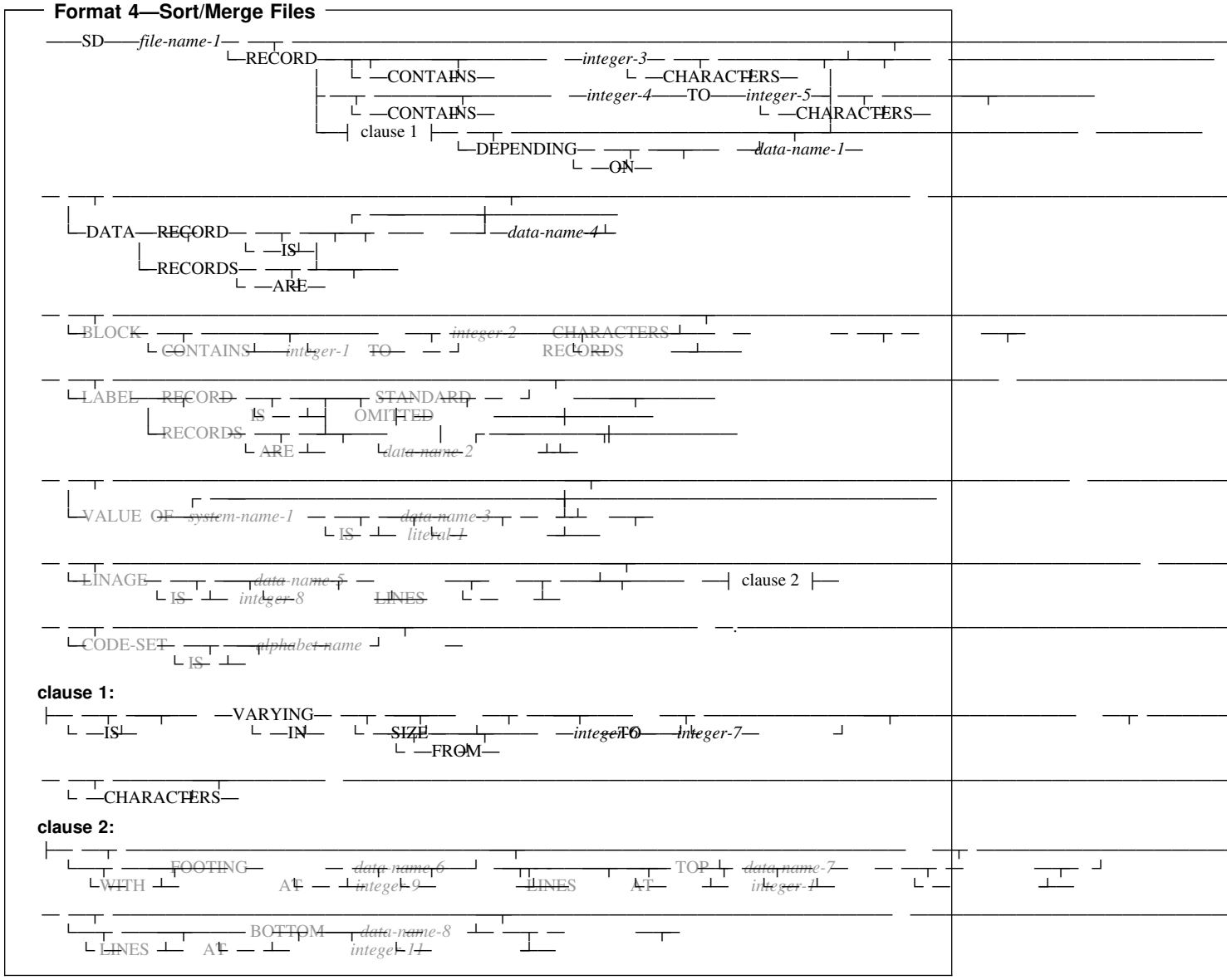
Format 2—Relative/Indexed Files



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.

EXTERNAL Clause

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.

 1   SORT-RECORD   PICTURE  X(8 ).
```

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

Workstation Under AIX, OS/2, and Windows, the BLOCK CONTAINS clause is not supported for line sequential files. It is treated as a comment for sequential, relative and indexed files. Workstation

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 example, if you have a block with 10 DBCS characters, the BLOCK CONTAINS clause should say `BLOCK CONTAINS 2 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.

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.

BLOCK CONTAINS Clause

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.

When running under OS/390, BLOCK CONTAINS 0 can be specified for QSAM files; the block size is determined at run time from the DD parameters or the data set label.

If the RECORD CONTAINS 0 CHARACTERS clause is specified, and the BLOCK CONTAINS 0 CHARACTERS clause is specified (or omitted), the block size is determined at run time from the DD parameters or the data set label of the file. For output data sets, with either of the above conditions, the DCB used by Language Environment will have a zero block size value. If you do not specify a block size value, the operating system might select a System Determined Block Size (SDB). See the operating system specifications for further information on SDB.

BLOCK CONTAINS can be omitted for SYSIN/SYSOUT files under OS/390. The blocking 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 run 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 run 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 2 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 209 and “SYNCHRONIZED Clause” on page 202.)

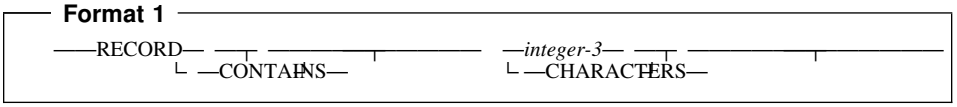
When the RECORD clause is omitted, the compiler determines the record lengths from the record descriptions. When one of the entries within a record description contains an OCCURS DEPENDING ON clause, the compiler uses the maximum value of the variable-length item to calculate the number of character 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.

Format 1



integer-3

Must be an unsigned integer that specifies the number of character positions contained in each record in the file.

RECORD Clause

Workstation 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. Workstation

Host Under OS/390, 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. Host

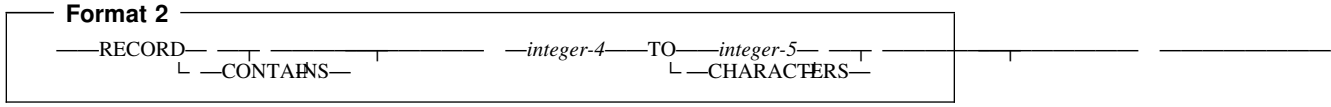
Do not specify the RECORD CONTAINS 0 clause for an SD entry.

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.

Workstation Under AIX, OS/2, and Windows, the Format 2 RECORD clause is not supported for line sequential files. Workstation

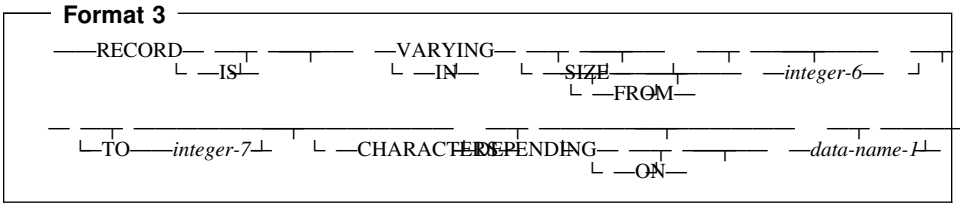
Format 2



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.

Data-name-1 cannot be a windowed date field.

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

Workstation Under AIX, OS/2, and Windows, the LABEL RECORDS clause is not supported 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 Workstation

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.

data-name-2
User labels are present in addition to standard labels. Data-name-2 specifies the name of a user label record. Data-name-2 must appear as the subject of a record description entry associated with the file.

The LABEL RECORDS clause is treated as a comment under an SD.

VALUE OF Clause

Workstation Under AIX, OS/2, and Windows, the VALUE OF clause is not supported for line sequential files. It is treated as a comment for sequential, relative and indexed files. Workstation

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.

The VALUE OF clause is treated as a comment under an SD.

DATA RECORDS Clause

Workstation Under AIX, OS/2, and Windows, the DATA RECORDS clause is not supported for line sequential files. It is treated as a comment for sequential, relative and indexed files. Workstation

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.

As an IBM extension, the data-name need not have an 01 level number record 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 files opened OUTPUT 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.

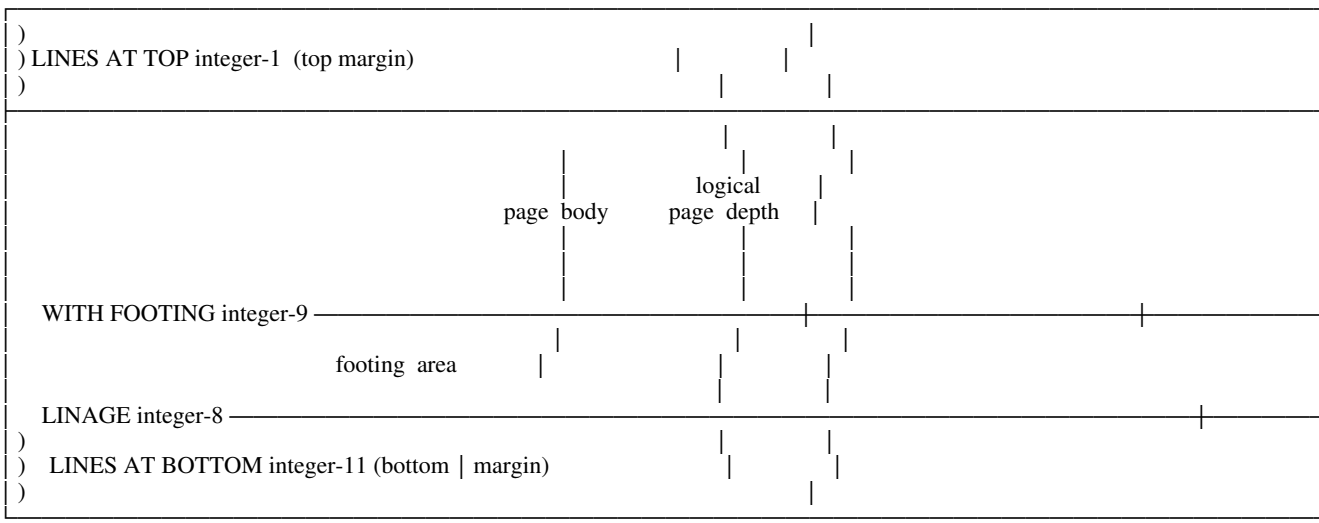


Figure 5. LINAGE Clause Phrases

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 156 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 437 for the behavior of carriage control characters in EXTERNAL files.

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 13.

RECORDING MODE Clause

Under OS/390 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 Clause

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 COBOL for OS/390 & 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 OS/390 & 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

Workstation Under AIX, OS/2, and Windows the CODE-SET clause is not supported for line sequential files. It is treated as a comment for sequential, relative and indexed files. Workstation

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.

CODE-SET Clause

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.

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.

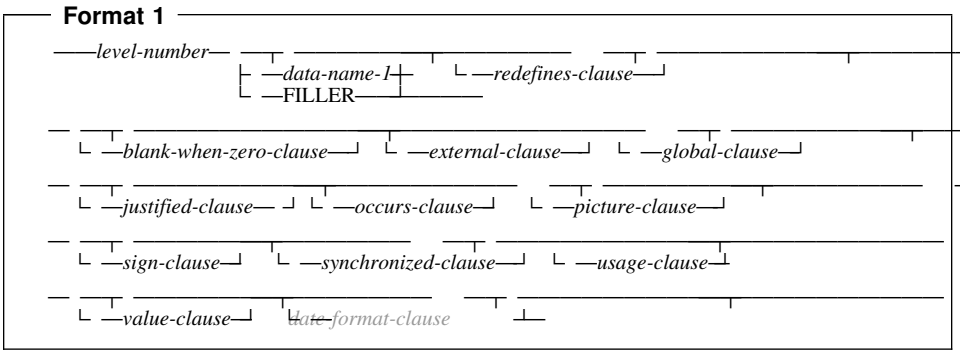
This chapter describes the coding of data description entries and record description entries (which are sets of data description entries). The single term **data description entry** is used in this chapter to refer to data and record description entries.

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.

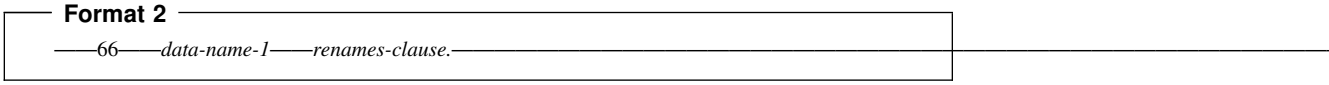
© Copyright IBM Corp. 1991, 1998

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.

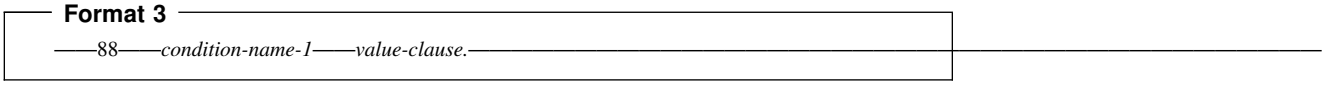
All level-66 entries associated with one record must immediately follow the last data description entry in that record.

Details are contained in “RENAMES Clause” on page 198.

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.

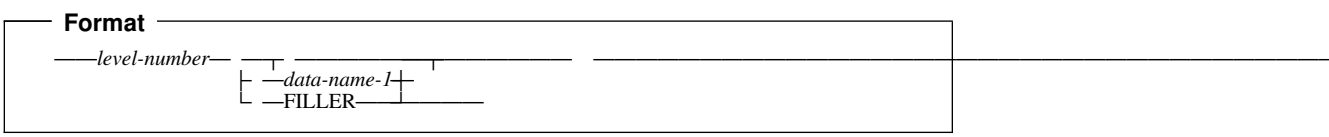
A **conditional variable** is a data item that can assume one or more values, that can, in turn, be associated with a condition-name.

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 217.

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 137

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.

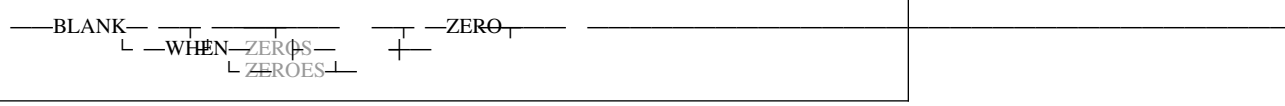
In a MOVE CORRESPONDING statement, or in an ADD CORRESPONDING or SUBTRACT CORRESPONDING statement, FILLER items are ignored. In an INITIALIZE statement, elementary FILLER items are ignored.

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
Date fields
DBCS items
External or internal floating-point items
Items described with USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE

DATE FORMAT Clause

The DATE FORMAT clause specifies that a data item is a windowed or expanded date field:

Windowed date fields
Contain a windowed (2-digit) year, specified by a DATE FORMAT clause containing YY.

Expanded date fields
Contain an expanded (4-digit) year, specified by a DATE FORMAT clause containing YYYY.

If the NODATEPROC compiler option is in effect, the DATE FORMAT clause is syntax checked, but has no effect on the execution of the program. NODATEPROC disables date processing. The rules and restrictions described in this reference for the DATE FORMAT clause and date fields apply only if the DATEPROC compiler option is in effect.

Format

The *date-pattern* is a character string, such as YYXXXX, representing a windowed or expanded year optionally followed or preceded by one through four characters representing other parts of a date, such as the month and day:

Date-pattern string...	Specifies that the data item contains...
YY	A windowed (2-digit) year.
YYYY	An expanded (4-digit) year.
X	A single character; for example, a digit representing a semester or quarter (1–4).
XX	Two characters; for example, digits representing a month (01–12).
XXX	Three characters; for example, digits representing a day of the year (001–366).
XXXX	Four characters; for example, 2 digits representing a month (01–12) and 2 digits representing a day of the month (01–31).

For an introduction to date fields and related terms, see “Millennium Language Extensions and Date Fields” on page 58. For details on using date fields in applications, see the *IBM COBOL Programming Guide* for your platform, or the *IBM COBOL Millennium Language Extensions Guide*.

Semantics of Windowed Date Fields

Windowed date fields undergo automatic expansion relative to the century window when they are used as operands in arithmetic expressions or arithmetic statements. However, the result of incrementing or decrementing a windowed date is still treated as a windowed date for further computation, comparison, and storing.

When used in the following situations, windowed date fields are treated as if they were converted to expanded date format:

- Operands in subtractions in which the other operand is an expanded date
- Operands in relation conditions
- A sending field in arithmetic or MOVE statements

The details of the conversion to expanded date format depend on whether the windowed date field is numeric or alphanumeric.

Given a century window starting year of 19*nn*, the year part (*yy*) of a numeric windowed date field is treated as if it was expanded as follows:

- If *yy* is less than *nn*, then add 2000 to *yy*

If *yy* is equal to or greater than *nn*, then add 1900 to *yy*

For signed numeric windowed date fields, this means that there can be two representations of some years. For instance, windowed year values 99 and -01 are both treated as 1999, since 1900 + 99 = 2000 + -01.

Alphanumeric windowed date fields are treated in a similar manner, but using a prefix of “19” or “20” instead of adding 1900 or 2000.

For example, when used as an operand of a relation condition, a windowed date field defined by:

```
1 DATE-FIELD  DATE FORMAT YYXXXX PICTURE 9(6)
                VALUE IS 45 1 1.
```

is treated as if it was an expanded date field with a value of:

19450101, if the century window starting year is 1945 or earlier

or

20450101, if the century window starting year is later than 1945

Date Trigger Values (Host Only)

Host When the DATEPROC(TRIG) compiler option is in effect, expansion of windowed date fields is sensitive to certain trigger or limit values in the windowed date field.

For alphanumeric windowed date fields, these special values are LOW-VALUE, HIGH-VALUE, and SPACE. For alphanumeric and numeric windowed date fields with at least one X in the DATE FORMAT clause (that is, windowed date fields other than just a windowed year), values of all zeros or all nines are also treated as triggers.

The all-zero value is intended to act as a date earlier than any valid date. The purpose of the all-nines value is to behave like a date later than any valid date.

When a windowed date field contains a trigger in this way, it is expanded as if the trigger value were copied to the century part of the expanded date result, rather than inferring 19 or 20 as the century value.

This special trigger expansion is done when a windowed date field is used either as an operand in a relation condition or as the sending field in an arithmetic or MOVE statement. Trigger expansion is not done when windowed date fields are used as operands in arithmetic expressions, but may be applied to the final windowed date result of an arithmetic expression. Host

Restrictions On Using Date Fields

The following pages describe restrictions on using date fields in these contexts:

Combining the DATE FORMAT clause with other clauses
Group items consisting only of a date field
Language elements that treat date fields as non-dates
Language elements that do not accept date fields as arguments

For restrictions on using date fields in other contexts, see:

"Arithmetic with Date Fields" on page 235
"Date Fields" (in conditional expressions) on page 244
"ADD Statement" on page 282
"SUBTRACT Statement" on page 425
"MOVE Statement" on page 352

Combining the DATE FORMAT Clause with Other Clauses

The only phrases of the USAGE clause that can be combined with the DATE FORMAT clause are DISPLAY, COMPUTATIONAL (or its equivalents, COMPUTATIONAL-4 and BINARY), and COMPUTATIONAL-3 (or its equivalent, PACKED-DECIMAL). The DATE FORMAT clause is not allowed for USAGE COMP data items if the TRUNC(BIN) compiler option is in effect.

The PICTURE clause character-string must specify the same number of characters or digits as the DATE FORMAT clause. For alphanumeric date fields, the only PICTURE character-string symbols allowed are A, 9, and X, with at least one X. For numeric date fields, the only PICTURE character-string symbols allowed are 9 and S.

The following clauses are not allowed for a data item defined with DATE FORMAT:

BLANK WHEN ZERO
JUSTIFIED
SEPARATE CHARACTER phrase of the SIGN clause

The EXTERNAL clause is not allowed for a windowed date field or a group item containing a windowed date field subordinate item.

Some restrictions apply when combining the following clauses with DATE FORMAT:

REDEFINES (see page 195)
VALUE (see page 217)

Group Items That Are Date Fields

If a group item is defined with a DATE FORMAT clause, then the following restrictions apply:

The elementary items in the group must all be USAGE DISPLAY.

The length of the group item must be the same number of characters as the *date-pattern* in the DATE FORMAT clause.

If the group consists solely of a date field with USAGE DISPLAY, and both the group and the single subordinate item have DATE FORMAT clauses, then the DATE FORMAT clauses must be identical.

If the group item contains subordinate items that subdivide the group, then the following restrictions apply:

1. If a named (not FILLER) subordinate item consists of exactly the year part of the group item date field, and has a DATE FORMAT clause, then the DATE

FORMAT clause must be YY or YYYY, with the same number of year characters as the group item.

2. If the group item is a Gregorian date with a DATE FORMAT clause of YYXXXX, YYYYXXXX, XXXXYY, or XXXXYYYY, and a named subordinate date data item consists of the year and month part of the Gregorian date, then its DATE FORMAT clause must be YYXX, YYYYXX, XXYY, or XXYYYY, respectively (or, for a group date format of YYYYXXXX, a subordinate date format of YYXX as described below).
3. A windowed date field can be subordinate to an expanded date field group item if the subordinate item starts two characters after the group item, neither date is in year-last format, and the date format of the subordinate item either has no Xs, or has the same number of Xs following the Ys as the group item, or is YYXX under a group date format of YYYYXXXX.
4. The only subordinate items that can have a DATE FORMAT clause are those that define the year part of the group item, the windowed part of an expanded date field group item, or the year and month part of a Gregorian date group item, as discussed in the above restrictions.

For example, the following defines a valid group item:

```
1     YYMMDD     DATE  FORMAT  YYXXXX.
 2     YYMM       DATE FORMAT YYXX.
  3    YY         DATE FORMAT YY PICTURE 99.
  3                          PICTURE  99.
 2     DD                    PICTURE  99.
```

Language Elements That Treat Date Fields As Non-Dates

If date fields are used in the following language elements, they are treated as non-dates. That is, the DATE FORMAT is ignored, and the content of the date data item is used without undergoing automatic expansion.

In the Environment Division FILE-CONTROL paragraph:

SELECT ... ASSIGN USING data-name
SELECT ... PASSWORD IS data-name
SELECT ... FILE STATUS IS data-name

In Data Division entries:

LABEL RECORD IS data-name
LABEL RECORDS ARE data-name
LINAGE IS data-name FOOTING data-name TOP data-name BOTTOM data-name

In class conditions

In sign conditions

In DISPLAY statements

Language Elements That Do Not Accept Windowed Date Fields As Arguments

Windowed date fields cannot be used as:

Data-names in the following formats of the Environment Division FILE-CONTROL paragraph:

SELECT ... RECORD KEY IS
SELECT ... ALTERNATE RECORD KEY IS
SELECT ... RELATIVE KEY IS

A data-name in the RECORD IS VARYING DEPENDING ON clause of a Data Division File Description (FD) or Sort Description (SD) entry.

The object of an OCCURS DEPENDING ON clause of a Data Division data definition entry.

The key in an ASCENDING KEY or DESCENDING KEY phrase of an OCCURS clause of a Data Division data definition entry.

Any data-name or identifier in the following statements:

CANCEL
GO TO ... DEPENDING ON
INSPECT
SET
SORT
STRING
UNSTRING

In the CALL statement, as the identifier containing the program name.

In the INVOKE statement, as the identifier specifying the object on which the method is invoked, or the identifier containing the method name.

Identifiers in the TIMES and VARYING phrases of the PERFORM statement (windowed date fields *are* allowed in the PERFORM conditions).

An identifier in the VARYING phrase of a serial (format 1) SEARCH statement, or any identifier in a binary (format 2) SEARCH statement (windowed date fields *are* allowed in the SEARCH conditions).

An identifier in the ADVANCING phrase of the WRITE statement.

Arguments to intrinsic functions, except the UNDATE intrinsic function.

Workstation Under AIX, OS/2, and Windows, windowed date fields cannot be used as ascending or descending keys in MERGE or SORT statements. Workstation

Host Under OS/390 and VM, windowed date fields can be used as ascending or descending keys in MERGE and SORT statements, with some restrictions. For details, see “MERGE Statement” on page 345 and “SORT Statement” on page 407.
Host

GLOBAL Clause

Language Elements That Do Not Accept Date Fields As Arguments

Neither windowed date fields nor expanded date fields can be used:

- In the DIVIDE statement, except as an identifier in the GIVING or REMAINDER clause.
- In the MULTIPLY statement, except as an identifier in the GIVING clause.

(Date fields cannot be used as operands in division or multiplication.)

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-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.

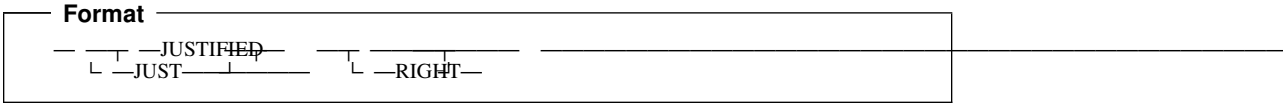
Two programs in a run unit can reference common data in the following circumstances:

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
For date fields
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.

OCCURS Clause

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 141).

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 209.

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.

Formats for the OCCURS clause include fixed-length tables or variable-length tables.

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:

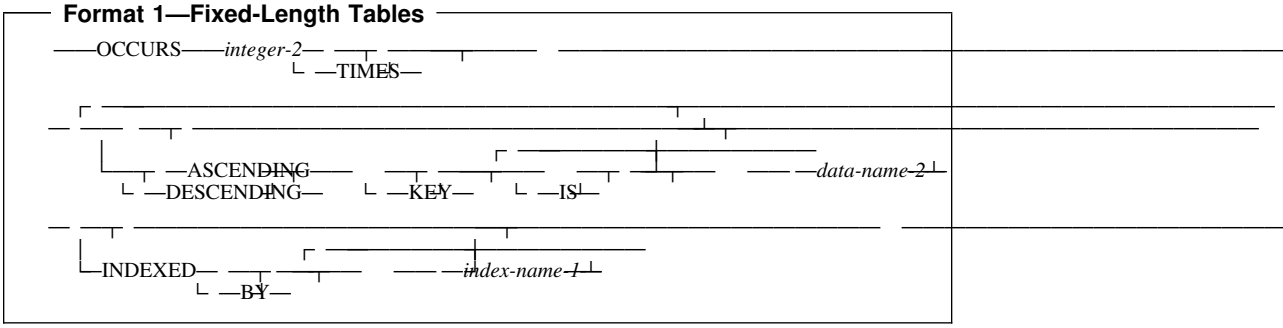
Has a level number of 01, 66, 77, or 88.

Describes a redefined data item. (However, a redefined item can be subordinate to an item containing an OCCURS clause.) See "REDEFINES Clause" on page 195.

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 243). 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. Data-name-2 cannot be a windowed date field. Data-name-2 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.

Host Under OS/390 and VM, a key can have COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, or COMPUTATIONAL-4 usage. Host

Workstation Under AIX, OS/2, and Windows, a key can have COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, COMPUTATIONAL-4, or COMPUTATIONAL-5 usage. Workstation

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.
 1  TABLE-RECORD.
   5  EMPLOYEE-TABLE  OCCURS  1    TIMES
         ASCENDING KEY IS WAGE-RATE EMPLOYEE-NO
         INDEXED BY A,    B.
     1    EMPLOYEE-NAME                              PIC  X(2 ).
     1    EMPLOYEE-NO                               PIC  9(6).
     1    WAGE-RATE                                 PIC  9999V99.
      1    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.

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 51.

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. However, in the following cases, indexes are allocated on a per-invocation basis. Thus, you must SET the value of the index on every entry for indexes on tables in the:

Local-Storage Section

Working-Storage Section of a class definition (object instance variables)

Linkage Section of a:

- Method
- Program compiled with the RECURSIVE attribute
- 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.

As an IBM extension, unreferenced index names need not be uniquely defined.

In one table entry, up to 12 index-names can be specified.

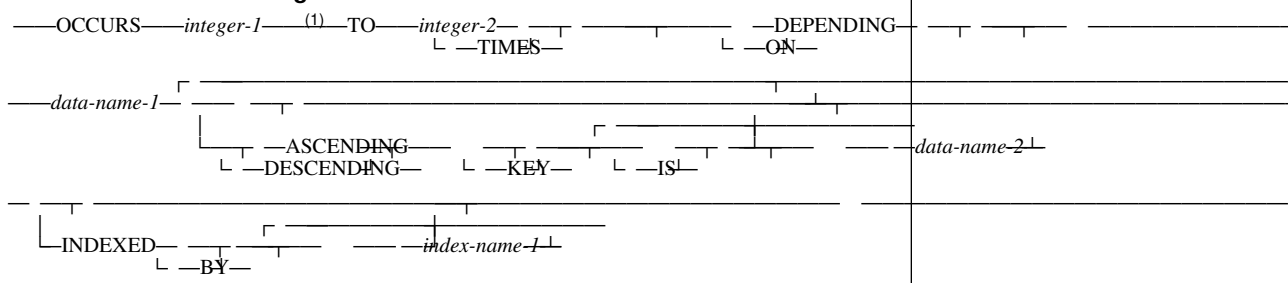
If a data item possessing the GLOBAL attribute includes a table accessed with an index, that index also possesses the GLOBAL attribute. Therefore, the scope of an index-name is identical to that of the data-name which names the table whose index is named by that index-name and the scope of name rules for data-names apply.

OCCURS DEPENDING ON Clause

Variable-Length Tables

Variable-length tables are specified using the OCCURS DEPENDING ON clause.

Format 2—Variable-Length Tables



Note:

[1] Integer-1 is optional as an IBM extension. If integer-1 is omitted, a value of 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.

The object of the OCCURS DEPENDING ON clause must describe an integer data item. The object cannot be a windowed date field.

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).

The object of the OCCURS DEPENDING ON clause may not be variably located; the object cannot follow an item that contains an OCCURS DEPENDING ON 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.

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.

PICTURE Clause

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.

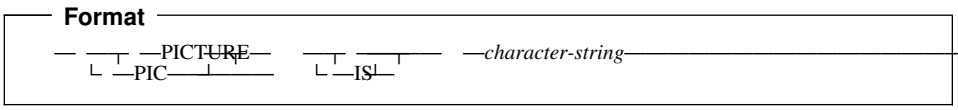
All data-names used in the OCCURS clause can be qualified; they can not be subscripted or indexed.

The ASCENDING/DESCENDING KEY and INDEXED BY clauses are described under "Fixed-Length Tables" on page 173.

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.

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 180. The sequence in which PICTURE clause symbols must be specified is shown in Figure 6 on page 183. 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

All other lowercase letters are not equivalent to their corresponding uppercase representations.

The heading **Size** refers to the number of bytes the symbol contributes to the actual size of the data item.

In the following description of the PICTURE clause, *cs* indicates any valid currency symbol. For details, see “Currency Symbol” on page 184.

PICTURE Clause

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	
B	For Non-DBCS data—a character position into which the space character is inserted.	Occupies 1 byte	
	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	
E	Marks the start of the exponent in an external floating-point item.	Occupies 1 byte	
G	A DBCS character position	Occupies 2 bytes	Cannot be specified for a non-DBCS item. 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 568.
N	A DBCS character position	Occupies 2 bytes	Cannot be specified for a non-DBCS item. 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 568.
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 184.	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. 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.

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. 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. 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.

PICTURE Clause

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. 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.	
cs	Currency symbol, representing a character position into which a currency sign value is placed. The default currency symbol is the dollar sign ($). For details, see "Currency Symbol" on page 184.	The first occurrence of a currency symbol adds the number of characters in the currency sign value to the size of the data item. Each subsequent occurrence adds one character to the size of the data item.	

Figure 6 on page 183 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}	CS	E	{Z *}	{Z *}	{+ -}	{+ -}	CS	CS	9	A X	S	V	P	P	G	N
NON-FLOATING INSERTION SYMBOLS	B ,	●	●	●	●	●	●			●		●	●	●	●	●	●	●	●		●		●	■	
	0	●	●	●	●	●	●			●		●	●	●	●	●	●	●	●		●		●		
	/	●	●	●	●	●	●			●		●	●	●	●	●	●	●	●		●		●		
	,	●	●	●	●	●	●			●		●	●	●	●	●	●	●			●		●		
	.	●	●	●	●		●			●		●		●		●		●							
	{+ -}																								
	{+ -}	●	●	●	●	●				●	■	●	●			●	●	●			●	●	●		
	{CR DB}	●	●	●	●	●				●		●	●			●	●	●			●	●	●		
	CS						●																		
	E				■	■												■			■				
FLOATING INSERTION SYMBOLS	{Z *}	●	●	●	●		●			●		●													
	{Z *}	●	●	●	●	●	●			●		●	●								●		●		
	{+ -}	●	●	●	●					●				●											
	{+ -}	●	●	●	●	●				●					●	●					●		●		
	CS	●	●	●	●		●									●									
	CS	●	●	●	●	●	●								●	●					●		●		
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.
■	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 *cs*, 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 by the symbol $, or by a single character specified either in the CURRENCY compiler option or in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division.

If the CURRENCY SIGN clause is specified, the CURRENCY and NOCURRENCY compiler options are ignored. If the CURRENCY SIGN clause is not specified and the NOCURRENCY compiler option is in effect, the dollar sign ($) is used as the default currency sign value and currency symbol. For more information about the CURRENCY SIGN clause, see “CURRENCY SIGN Clause” on page 96. For more information about the CURRENCY and NOCURRENCY compiler options, see the *IBM COBOL Programming Guide* for your platform.

A currency symbol may be repeated within the PICTURE character-string to specify floating insertion. Different currency symbols must not be used in the same PICTURE character-string.

Unlike all other PICTURE clause symbols, currency symbols are case-sensitive: for example, 'D' and 'd' specify different currency symbols.

A currency symbol may be used only to define a numeric-edited item with USAGE DISPLAY.

In the following description of the PICTURE clause, *cs* indicates any valid currency symbol.

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 / , + - *cs*
G N

At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, -, or *cs* must be present in a PICTURE string.

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
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
DBCS items
External floating-point items

PICTURE Clause

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. For numeric date fields, the PICTURE character-string can contain only the symbols 9 and S.

The number of digit positions must range from 1 through 18, inclusive. For numeric date fields, the number of digit positions must match the number of characters specified by the DATE FORMAT clause.

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	through 9999
S99	-99 through +99
S999V9	-999.9 through +999.9
PPP999	through .999
S999PPP	-1 through -999 and +1 through +999 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 figura-

tive 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, the BINARY compiler option is only applicable to OS/2 and Windows programs.)

Numeric-edited Items

The PICTURE character-string can contain the following symbols:

B P V Z 9 / , . + - CR DB *cs*

The combinations of symbols allowed are determined from the PICTURE clause symbol order allowed (see Figure 6 on page 183), and the editing rules (see "PICTURE Clause Editing" on page 190).

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 , . + - CR DB *cs*

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 /

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.

Workstation 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 568.)

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. Workstation

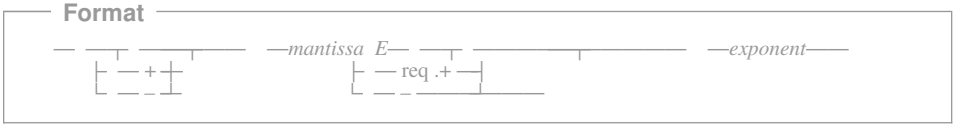
Other Clauses: When PICTURE clause symbol G is used, USAGE DISPLAY-1 must be specified.

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.

Each sign position occupies one byte of storage.

mantissa
The mantissa can contain the symbols:

9 . V

An actual decimal point can be represented with a period (.) while an assumed decimal point is represented by a V.

Either an actual or an assumed decimal point must be present in the mantissa; the decimal point can be leading, embedded, or trailing.

The mantissa can contain from 1 to 16 numeric characters.

E Indicates the exponent.

exponent
The exponent must consist of the symbol 99.

Other Clauses: The OCCURS, REDEFINES, RENAMES, and USAGE clauses may be associated with external floating-point items.

The SIGN clause is accepted as documentation and has no effect on the representation of the sign.

The SYNCHRONIZED clause is treated as documentation.

The following clauses are invalid with external floating-point items:

PICTURE Clause

BLANK WHEN ZERO
JUSTIFIED
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 in Table 13.

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	*cs* + – CR DB
	Floating insertion	*cs* + –
	Zero suppression	Z *
	Replacement	Z * + – *cs*
Alphanumeric	None	None
Alphanumeric-edited	Simple insertion	B 0 /
DBCS	Simple insertion	B
External floating-point	Special insertion	.

Simple Insertion Editing

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 within the item where the equivalent character is to be inserted. For edited DBCS items, each insertion symbol (B) is counted in the size of the item and represents the position within the item where the DBCS space is to be inserted.

For example:

PICTURE	Value of Data	Edited Result
X(1)/XX	ALPHANUMER 1	ALPHANUMER/ 1
X(5)BX(7)	ALPHANUMERIC	ALPHA NUMERIC
99,B999,B	1234	1, 234,
99,999	12345	12,345
GGBBGG	D1D2D3D4	D1D2 D3D4

Special Insertion Editing

This type of editing is valid for either numeric-edited items or external floating-point 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	1.23
999.99	12.34	12.34
999.99	123.45	123.45
999.99	1234.5	234.5
+999.99E+99	12345	+123.45E+ 2

Fixed Insertion Editing

This type of editing is valid only for numeric-edited items. The following insertion symbols are used:

cs

\+ − 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	$ 123.45
$9999.99DB	-123.45	$ 123.45DB

Floating Insertion Editing

This type of editing is valid only for numeric-edited items.

The following symbols are used:

cs + −

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 leftmost 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	$ .12
$,$$$,999.99	-1234.56	$1,234.56
+,+++,999.99	-123456.789	-123,456.78
$$,$$$,$$$.99CR	-1234567	$1,234,567. CR
++,+++,+++.+++	.	

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 * + – *cs*

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:

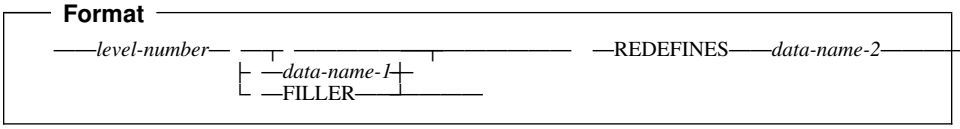
PICTURE	Value of Data	Edited Result
.	.	.
ZZZZ.ZZ	.	
ZZZZ.99	.	.
.99	.	.
ZZ99.99	.	.
Z,ZZZ.ZZ+	+123.456	123.45+
, . +	-123.45	123.45-
, , . +	+12345678.9	12,345,678.9 +
$Z,ZZZ,ZZZ.ZZCR	+12345.67	$ 12,345.67
$B , , . BBDB	-12345.67	$ 12,345.67 DB

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:

```
5    A PICTURE X(6).
5    B REDEFINES A.
 1   B-1                    PICTURE  X(2).
 1   B-2                    PICTURE  9(4).
5    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.

The data description entry for data-name-2, the redefined item, can contain a REDEFINES clause.

REDEFINES 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.

When the data item implicitly redefines multiple 01-level records in a file description (FD) entry, items subordinate to the redefining or redefined item can contain an 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:

```
5    A                    PICTURE  9999.
5     B REDEFINES A         PICTURE 9V999.
5     C REDEFINES A         PICTURE 99V99.
```

The redefining entry (identified by data-name-1), and any subordinate entries, must not contain any VALUE clauses.

An item described as USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE can be the subject or object of a REDEFINES clause.

An external or internal floating-point item can be the subject or object of a REDEFINES clause.

REDEFINES Clause Considerations

Data items within an area can be redefined without changing their lengths. For example:

```
5  NAME-2.
 1   SALARY                      PICTURE  XXX.
 1   SO-SEC-NO                   PICTURE  X(9).
 1   MONTH                       PICTURE  XX.
5   NAME-1 REDEFINES NAME-2.
 1   WAGE                        PICTURE  XXX.
 1   EMP-NO                      PICTURE  X(9).
 1   YEAR                        PICTURE  XX.
```

Data item lengths and types can also be re-specified within an area. For example:

```
5  NAME-2.
 1   SALARY                      PICTURE  XXX.
 1   SO-SEC-NO                   PICTURE  X(9).
 1   MONTH                       PICTURE  XX.
5   NAME-1 REDEFINES NAME-2.
 1   WAGE                        PICTURE  999V999.
 1   EMP-NO                      PICTURE  X(6).
 1   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:

```
5   B                          PICTURE   99 USAGE DISPLAY VALUE 8.
5   C REDEFINES B                PICTURE S99 USAGE COMPUTATIONAL-4.
5   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.

RENAMES Clause

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:

```
5  REGULAR-EMPLOYEE.
 1   LOCATION                          PICTURE  A(8).
 1   GRADE                             PICTURE  X(4).
 1   SEMI-MONTHLY-PAY                    PICTURE  9999V99.
 1   WEEKLY-PAY  REDEFINES  SEMI-MONTHLY-PAY
                                       PICTURE  999V999.

5  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
 1   LOCATION                          PICTURE  A(8).
 1   FILLER                            PICTURE  X(6).
 1   HOURLY-PAY                          PICTURE  99V99.
```

The REDEFINES clause can also be specified for an item subordinate to a redefining item. For example:

```
5  REGULAR-EMPLOYEE.
 1   LOCATION                          PICTURE  A(8).
 1   GRADE                             PICTURE  X(4).
 1   SEMI-MONTHLY-PAY                    PICTURE  999V999.

5  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
 1   LOCATION                          PICTURE  A(8).
 1   FILLER                            PICTURE  X(6).
 1   HOURLY-PAY                          PICTURE  99V99.
 1   CODE-H  REDEFINES  HOURLY-PAY       PICTURE  9999.
```

Undefined Results

Undefined results can occur when:

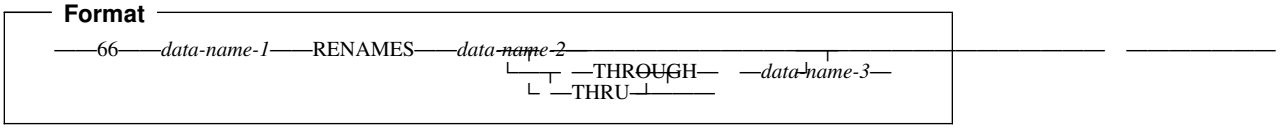
A redefining item is moved to a redefined item (that is, if B REDEFINES C and the statement MOVE B TO C is executed).

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



RENAMES Clause

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.

Can specify a DBCS data item if data-name-2 specifies a DBCS data item and the 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 rightmost 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.

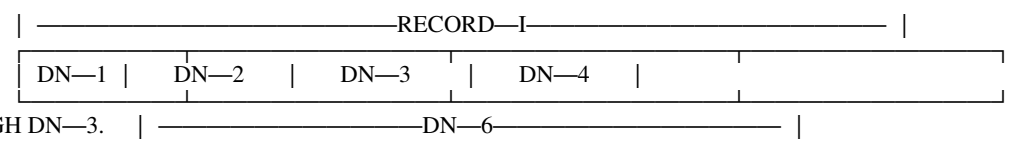
Figure 7 illustrates valid and invalid RENAMES clause specifications.

SIGN Clause

COBOL Specifications **Storage Layouts**

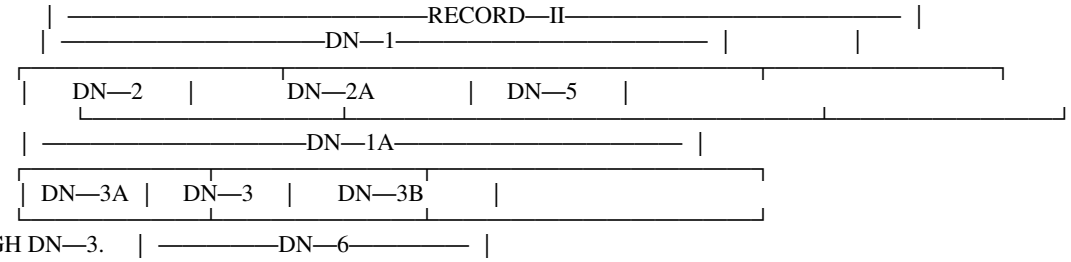
Example 1 (Valid)

```
 1  RECORD—I.
      5 DN—1... .
      5 DN—2... .
      5 DN—3... .
      5 DN—4... .
66    DN—6 RENAMES DN—1 THROUGH DN—3.
```



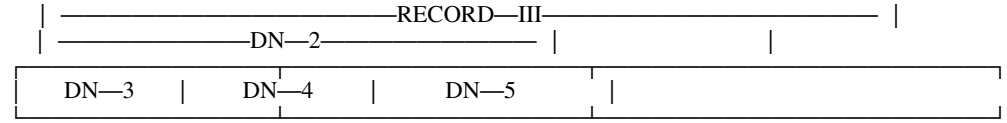
Example 2 (Valid)

```
 1  RECORD—II.
      5  DN—1.
         1  DN—2... .
         1  DN—2A... .
      5 DN—1A REDEFINES DN—1.
         1  DN—3A... .
         1  DN—3... .
         1  DN—3B... .
      5 DN—5... .
66    DN—6 RENAMES DN—2 THROUGH DN—3.
```



Example 3 (Invalid)

```
 1  RECORD—III.
      5  DN—2.
         1  DN—3... .
         1  DN—4... .
      5 DN—5... .
66    DN—6 RENAMES DN—2 THROUGH DN—3.  DN—6 is indeterminate
```



Example 4 (Invalid)

```
 1  RECORD—IV.
      5  DN—1.
         1  DN—2A... .
         1  DN—2B... .
         1  DN—2C REDEFINES DN—2B.
            15 DN—2... .
            15 DN—2D... .
      5 DN—3... .
66    DN—4 RENAMES DN—1 THROUGH DN—2.  DN—4 is indeterminate
```

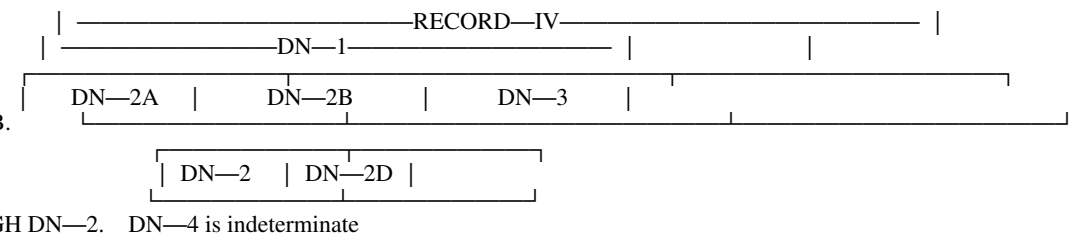
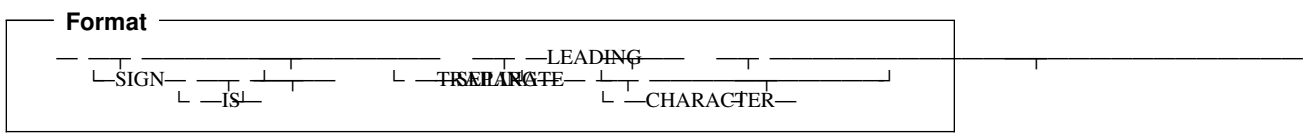


Figure 7. RENAMES Clause—Valid and Invalid Specifications

SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign for a numeric entry.

Format



SIGN Clause

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.

The SEPARATE CHARACTER phrase cannot be specified for a date field.

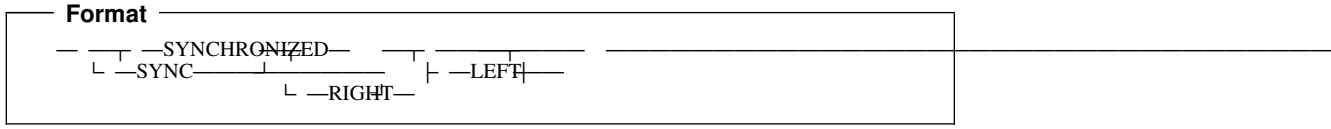
Every numeric data description entry whose PICTURE contains the symbol S is a signed numeric data description entry. If the SIGN clause is also specified for such an entry, and conversion is necessary for computations or comparisons, the conversion takes place automatically.

The SIGN clause is treated as documentation for external floating-point items. For internal floating-point items, the SIGN clause must not be specified.

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.

The SYNCHRONIZED clause can appear at the elementary level or at the group level (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.

Table 14 lists the effect of the SYNCHRONIZE clause on other language elements.

Table 14 (Page 1 of 2). SYNCHRONIZE Clause Effect 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.

Table 14 (Page 2 of 2). SYNCHRONIZE Clause Effect on Other Language Elements

Language Element	Comments
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. When the synchronized clause is not specified for a subordinate data item (one with a level number of 02 through 49): 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). 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. 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:
	`2 A PICTURE X(4).`
	`2 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

m = 4 for binary items of 5-digit length or more: USAGE IS INDEX, USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, USAGE IS OBJECT REFERENCE, and COMPUTATIONAL-1 data items

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:

```
1  FIELD-A.
  5  FIELD-B                          PICTURE  X(5).
  5  FIELD-C.
    1    FIELD-D                       PICTURE  XX.
   [1    SLACK-BYTES                     PICTURE X.    INSERTED BY COMPILER]
    1    FIELD-E   COMPUTATIONAL   PICTURE  S9(6)  SYNC.

1  FIELD-L.
  5  FIELD-M                          PICTURE  X(5).
  5  FIELD-N                          PICTURE  XX.
[ 5  SLACK-BYTES                       PICTURE X.    INSERTED BY COMPILER]
  5  FIELD-O.
    1    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:

```
1 WORK-RECORD.
  5  WORK-CODE                        PICTURE  X.
  5  COMP-TABLE  OCCURS  1   TIMES.
    1    COMP-TYPE                    PICTURE  X.
   [1    SLACK-BYTES                   PIC XX.    INSERTED BY COMPILER]
    1    COMP-PAY                      PICTURE S9(4)V99 COMP SYNC.
    1    COMP-HRS                      PICTURE S9(3) COMP SYNC.
    1    COMP-NAME                      PICTURE  X(5).
```

SYNCHRONIZED Clause

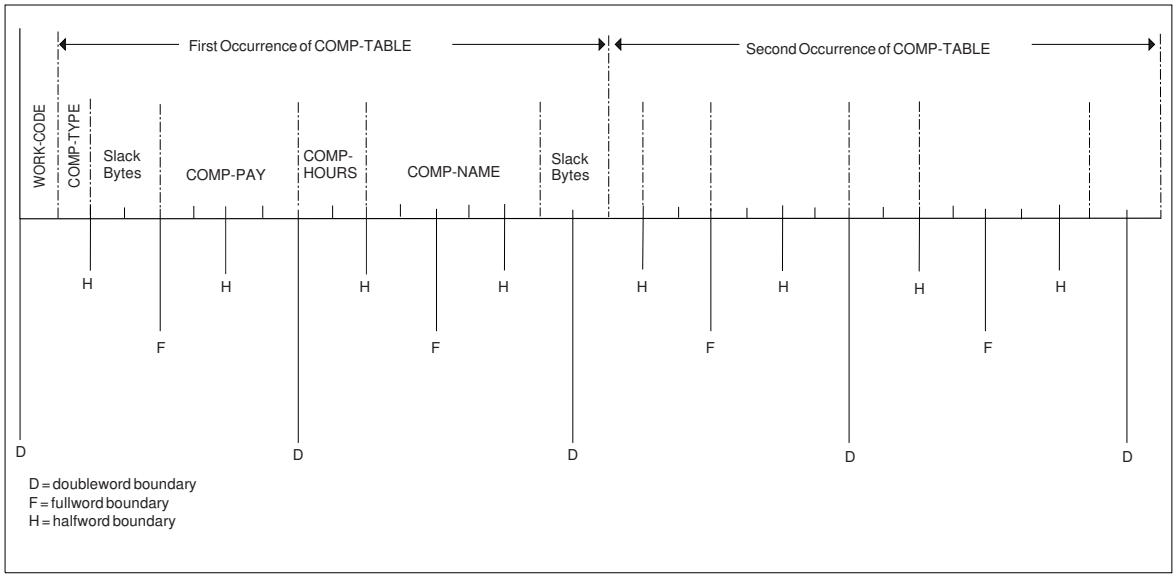


Figure 8. Insertion of Slack Bytes within a Record

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:

```
 1  WORK-RECORD.
   5  WORK-CODE                    PICTURE  X.
   5  COMP-TABLE  OCCURS  1   TIMES.
     1    COMP-TYPE                PICTURE  X.
    [1    SLACK-BYTES              PIC XX.   INSERTED BY COMPILER ]
     1    COMP-PAY                 PICTURE S9(4)V99 COMP SYNC.
     1    COMP-HRS                 PICTURE  S9(3)   COMP  SYNC.
     1    COMP-NAME                 PICTURE  X(5).
    [1    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.

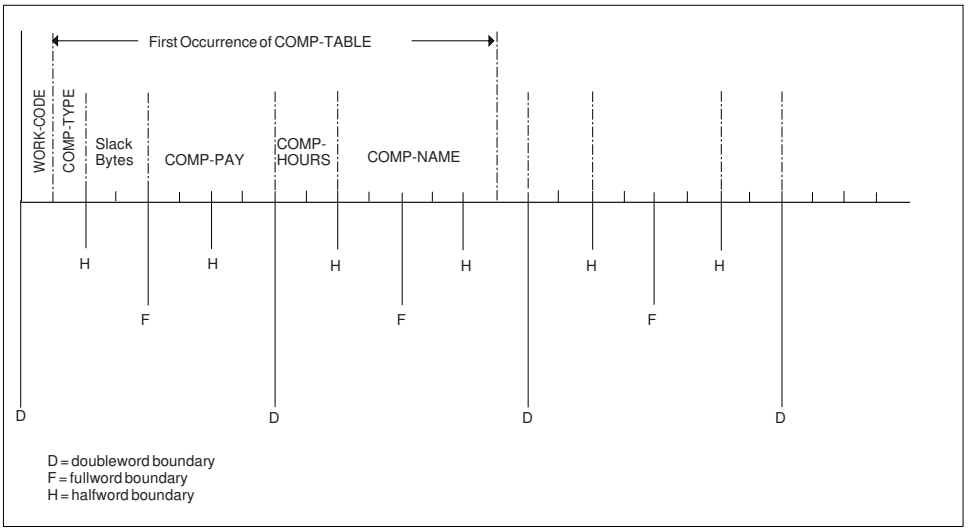


Figure 9. Insertion of Slack Bytes between Records

Each succeeding occurrence within the table will now begin at the same relative position as the first.

Slack Bytes between Records

Host Under OS/390 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. Host

The lengths of all the elementary data items in the record, including all slack bytes, are added. (For variable-length records under OS/390 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:

```
1  COMP-RECORD.
   5  A-1              PICTURE  X(5).
   5  A-2              PICTURE  X(3).
   5  A-3              PICTURE  X(3).
   5  B-1              PICTURE  S9999   USAGE  COMP  SYNCHRONIZED.
   5  B-2              PICTURE  S99999  USAGE  COMP  SYNCHRONIZED.
   5  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:

```
 1  COMP-RECORD.
   5  A-1                PICTURE  X(5).
   5  A-2                PICTURE  X(3).
   5  A-3                PICTURE  X(3).
 [ 5  SLACK-BYTE-1        PICTURE X.     INSERTED BY COMPILER]
   5  B-1                PICTURE  S9999  USAGE  COMP  SYNCHRONIZED.
 [ 5  SLACK-BYTE-2        PICTURE XX.    INSERTED BY COMPILER]
   5  B-2                PICTURE  S99999  USAGE  COMP  SYNCHRONIZED.
   5  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.

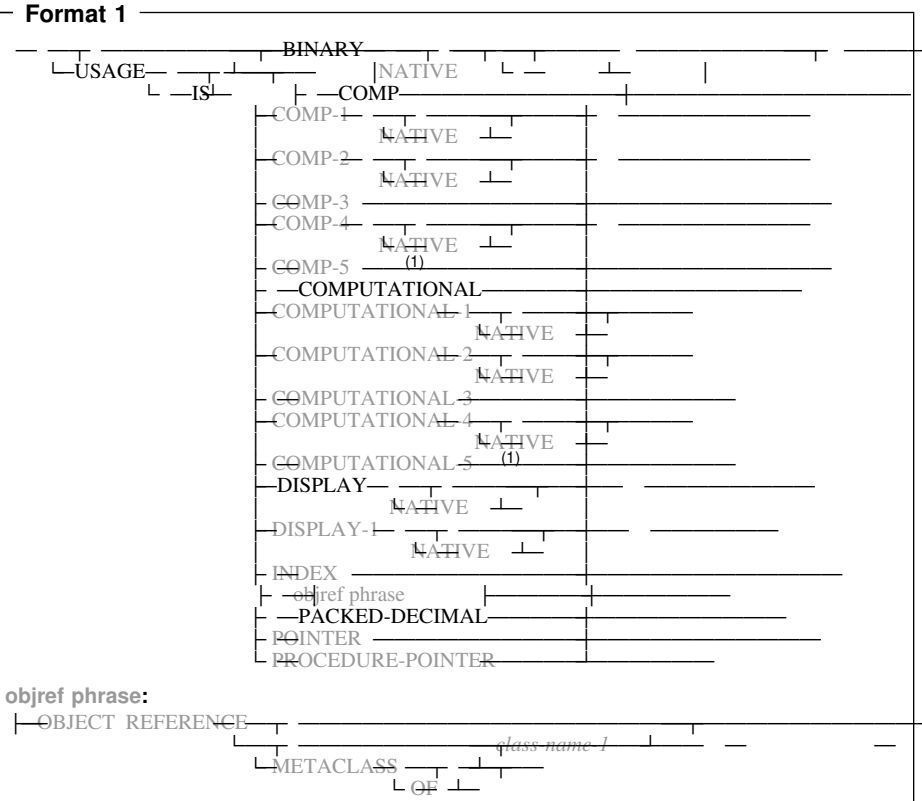
The final record description entry appears as:

```
 1  COMP-RECORD.
   5  A-1                PICTURE  X(5).
   5  A-2                PICTURE  X(3).
   5  A-3                PICTURE  X(3).
 [ 5  SLACK-BYTE-1        PICTURE X.     INSERTED BY COMPILER]
   5  B-1                PICTURE  S9999  USAGE  COMP  SYNCHRONIZED.
 [ 5  SLACK-BYTE-2        PICTURE XX.    INSERTED BY COMPILER]
   5  B-2                PICTURE  S99999  USAGE  COMP  SYNCHRONIZED.
   5  B-3                PICTURE  S9999   USAGE  COMP  SYNCHRONIZED.
   5  FILLER              PICTURE  XX.  [SLACK  BYTES  YOU  ADD]
```

USAGE Clause

The USAGE clause specifies the format of a data item in computer storage.

Format 1



objref phrase:

Note:

1 COMP-5 and COMPUTATIONAL-5 are only supported on AIX, OS/2, and Windows.

The NATIVE phrase is treated as a comment for COMP-3, COMPUTATIONAL-3, COMP-5, COMPUTATIONAL-5, and PACKED-DECIMAL data items. NATIVE has no effect on OS/390 and VM.

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.

USAGE Clause

When the USAGE clause is not specified at either the group or elementary level, it is assumed that the usage is DISPLAY.

For data items defined with the DATE FORMAT clause, only usage DISPLAY and COMP-3 (or its equivalents, COMPUTATIONAL-3 and PACKED-DECIMAL) are allowed. For details, see "Combining the DATE FORMAT Clause with Other Clauses" on page 167.

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

COMPUTATIONAL-1 and COMPUTATIONAL-2 items (internal floating-point) cannot 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 OS/390 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.

Note: BINARY, COMPUTATIONAL, and COMPUTATIONAL-4 data items can be affected by the BINARY and TRUNC compiler option specifications. For information on the effect of these compiler options, see the *IBM COBOL Programming Guide* for your platform. (The BINARY compiler option is only applicable to OS/2 and Windows 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.

COMPUTATIONAL-1 or COMP-1 (Floating-Point)
Specified for internal floating-point items (single precision). COMP-1 items are 4 bytes long.

Workstation COMP-1 data items are affected by the FLOAT(NATIVE|HEX) compiler option. For details, see the *IBM COBOL Programming Guide* for your platform Workstation

COMPUTATIONAL-2 or COMP-2 (Long Floating-Point)
Specified for internal floating-point items (double precision). COMP-2 items are 8 bytes long.

Workstation COMP-2 data items are affected by the FLOAT(NATIVE|HEX) compiler option. For details, see the *IBM COBOL Programming Guide* for your platform Workstation

COMPUTATIONAL-3 or COMP-3 (Internal Decimal)
This is the equivalent of PACKED-DECIMAL.

COMPUTATIONAL-4 or COMP-4 (Binary)
This is the equivalent of BINARY.

COMPUTATIONAL-5 or COMP-5 (Native Binary)
Workstation Under AIX, OS/2, and Windows, this represents native binary data. In a COMP-5 binary data item, the total precision of the literal (integer and decimal positions) can be up to the maximum value that can be accommodated in the number of bytes allocated for the binary field (for example, 2, 4, or 8 bytes) rather than conforming to the maximum integer and decimal positions specified in the PICTURE clause. This is always the native binary data, independent of the BINARY compiler option. Workstation

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

USAGE Clause

External floating-point
External decimal (numeric)

Alphabetic, **alphanumeric**, **alphanumeric-edited**, and **numeric-edited** items are discussed in “Data Categories and PICTURE Rules” on page 185.

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.

Effect of CHAR(EBCDIC) Compiler Option (Workstation Only): Character data items are treated as EBCDIC when the CHAR(EBCDIC) option is used, unless the character data is defined with the NATIVE phrase. Also note, group items are affected by the CHAR options as well. A group item is treated as a USAGE DISPLAY item and consists of either native single byte characters (with CHAR(NATIVE)) or EBCDIC characters (with CHAR(EBCDIC)). Any USAGE clause specified on a group applies to the elementary items within the group and not to the group itself for the purpose of defining semantics involving group items.

Command-line arguments are always passed in as native data types. If you specify the host data type compiler options (CHAR(EBCDID), FLOAT(HEX), or BINARY(S390)), you must specify the NATIVE phrase on any arguments with data types affected by these compiler options.

DISPLAY-1 Phrase

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.

An index data item can be referred to directly in the USING phrase of an ENTRY statement.

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 DATE FORMAT, 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.

SYNCHRONIZED can be used with USAGE IS INDEX to obtain efficient use of the index data item.

OBJECT REFERENCE Phrase

A data item defined with the OBJECT REFERENCE phrase is an **object reference**.

class-name-1

An optional class name.

You must declare class-name-1 in the REPOSITORY paragraph in the Configuration Section of the containing class or outermost program. If specified, class-name-1 indicates that data-name always refers to an object-instance of class 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.

USAGE Clause

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 DATE FORMAT, 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 DATE FORMAT, 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.

Host Under OS/390 and VM, a procedure-pointer data item is an 8-byte elementary item. Host

Workstation Under AIX, OS/2, and Windows, a procedure-pointer data item is a 4-byte elementary item. Workstation

The entry point for a procedure-pointer data item can be:

USAGE Clause

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.

The GLOBAL, EXTERNAL, and OCCURS clause can be used with USAGE IS PROCEDURE-POINTER.

A VALUE clause for a procedure-pointer data item can contain only NULL or NULLS.

The DATE FORMAT, JUSTIFIED, PICTURE, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items defined with the USAGE IS PROCEDURE-POINTER clause.

A procedure-pointer data item cannot be a conditional variable.

A procedure-pointer data item does not belong to any class or category.

Procedure-pointer data items are ignored in CORRESPONDING operations.

NATIVE Phrase

Host
Host

Under OS/390 and VM, the NATIVE phrase is treated as a comment.

Using the NATIVE phrase, you can mix characters, floating point, and binary data as represented on the S390 and native platform. The NATIVE phrase overrides the CHAR(EBCDIC), FLOAT(HEX), and BINARY(S390) compiler options, which indicate host data type usages. (Note, the BINARY compiler option is only applicable to OS/2 and Windows programs.)

Using both host and native data types within a program (ASCII and EBCDIC, Hex Floating point and IEEE floating point, and/or big endian and little endian binary) is only valid for those data items specifically defined with the NATIVE phrase.

Specifying NATIVE does not change the class or the category of the data item.

Numeric data items are treated in arithmetic operations (numeric comparisons, arithmetic expressions, assignment to numeric targets, arithmetic statement) based on their logical numeric values, regardless of their internal representations.

Characters are converted to the representation of the target item prior to an assignment.

Comparisons are done based on the collating sequence rules applicable to the operands. If native and non-native characters are compared, the comparison is based on the COLLSEQ option in effect.

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.

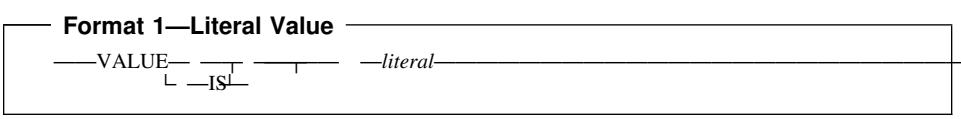
In the class Working-Storage Section, the VALUE clause can only be used in condition-name entries.

As an IBM extension, in the File and Linkage Sections, if the VALUE clause is used in 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



VALUE Clause

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.

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.

Any VALUE clause associated with COMPUTATIONAL-1 or COMPUTATIONAL-2 (internal floating-point) items must specify a floating-point literal. The condition-name VALUE phrase must also specify a floating-point literal. In addition, the figurative constant ZERO and both integer and decimal forms of the zero literal can be specified in a floating-point VALUE clause or condition-name VALUE phrase.

For information on floating-point literal values, see “Rules for Floating-point Literal Values:” on page 24.

A VALUE clause cannot be specified for external floating-point items.

A VALUE clause associated with a DBCS item must contain a DBCS literal or the figurative constant SPACE.

A data item cannot contain a VALUE clause if the prior data item contains a OCCURS clause with the DEPENDING ON phrase.

Rules for Literal Values:

Wherever a literal is specified, a figurative constant can be substituted.

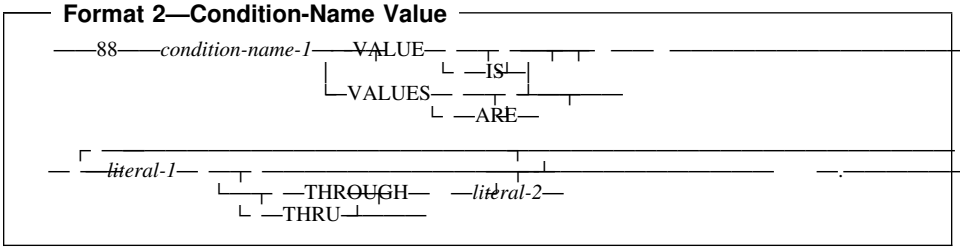
If the item is numeric, all VALUE clause literals must be numeric. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves, with one additional restriction: The literal must not have a value that requires truncation of nonzero digits. 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 239).

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, unless the associ-

ated data item is a non-year-last windowed date field. For details, see "Rules for Condition-Name Values:" on page 220.

In the VALUE clause of a data description entry (Format 2), all the literals specified for the THROUGH phrase must be DBCS literals if the associated conditional variable is a DBCS data item. The figurative constants SPACE and SPACES can be used as DBCS literals.

Host Under OS/390 and VM, the range of DBCS literals specified for the THROUGH phrase is based on the binary collating sequence of the hexadecimal values of the DBCS characters. Host

Workstation Under AIX, OS/2, and Windows, the range of nonnumeric literals or DBCS literals specified for the THROUGH phrase is based on the collating sequence indicated by the locale (except for single-byte character comparisons when a non-NATIVE collating sequence is in effect). For more information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 568. Workstation

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.

The conditional variable cannot be an item with USAGE IS POINTER, USAGE IS 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.

  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.

The VALUE clause is allowed for internal floating-point data items.

The VALUE clause is allowed for DBCS data items. Relation tests for DBCS data items are performed according to the rules for comparison of DBCS items. These rules can be found in "Comparison of DBCS Operands" on page 253.

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.

```
5  CITY-COUNTY-INFO.
   88   BRONX                VALUE  " 3NYC".
   88   BROOKLYN              VALUE  "24NYC".
   88   MANHATTAN              VALUE  "31NYC".
   88   QUEENS               VALUE  "41NYC".
   88   STATEN-ISLAND         VALUE  "43NYC".
 1  COUNTY-NO                PICTURE  99.
      88   DUTCHESS           VALUE  14.
      88   KINGS              VALUE  24.
      88   NEW-YORK            VALUE  31.
      88   RICHMOND            VALUE  43.
 1  CITY                    PICTURE  X(3).
      88   BUFFALO            VALUE  "BUF".
      88   NEW-YORK-CITY        VALUE  "NYC".
      88   POUGHKEEPSIE        VALUE  "POK".
5  POPULATION...
```

VALUE Clause

If the item is a windowed date field, the following restrictions apply:

- For alphanumeric conditional variables:
  - Both literal-1 and literal-2 (if specified) must be alphanumeric literals of the same length as the conditional variable.
  - The literals must not be specified as figurative constants.
  - If literal-2 is specified, then both literals must contain only decimal digits.
- If the YEARWINDOW compiler option is specified as a negative integer, then literal-2 must not be specified.
- If literal-2 is specified, then literal-1 must be less than literal-2 after applying the century window specified by the YEARWINDOW compiler option. That is, the expanded date value of literal-1 must be less than the expanded date value of literal-2.

For more information on using condition-names with windowed date fields, see "Condition-Name Conditions and Windowed Date Field Comparisons" on page 243.

Format 3

Format 3—NULL Value

This format assigns an invalid address as the initial value of an item defined as USAGE IS POINTER or USAGE IS PROCEDURE-POINTER. It also assigns an invalid object reference as the initial value of an item defined as USAGE IS OBJECT REFERENCE.

VALUE IS NULL can only be specified for elementary items described implicitly or explicitly as USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE.

Part 6. Procedure Division

Procedure Division Structure 225
Requirements for a Method Procedure Division 226
The Procedure Division Header 227
Declaratives 230
Procedures 231
Arithmetic Expressions 233
Conditional Expressions 239
Statement Categories 261
Statement Operations 264

Procedure Division Statements 277
ACCEPT Statement 277
ADD Statement 282
ALTER Statement 285
CALL Statement 287
CANCEL Statement 294
CLOSE Statement 296
COMPUTE Statement 300
CONTINUE Statement 302
DELETE Statement 303
DISPLAY Statement 305
DIVIDE Statement 309
ENTRY Statement 312
EVALUATE Statement 313
EXIT Statement 317
EXIT METHOD Statement 318
EXIT PROGRAM Statement 319
GOBACK Statement 320
GO TO Statement 321
IF Statement 323
INITIALIZE Statement 325
INSPECT Statement 328
INVOKE Statement 337
MERGE Statement 345
MOVE Statement 352
MULTIPLY Statement 357
OPEN Statement 359
PERFORM Statement 365
READ Statement 376
RELEASE Statement 385
RETURN Statement 387
REWRITE Statement 389
SEARCH Statement 393
SET Statement 400
SORT Statement 407

© Copyright IBM Corp. 1991, 1998

START Statement 415
STOP Statement 418
STRING Statement 420
SUBTRACT Statement 425
UNSTRING Statement 428
WRITE Statement 436

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

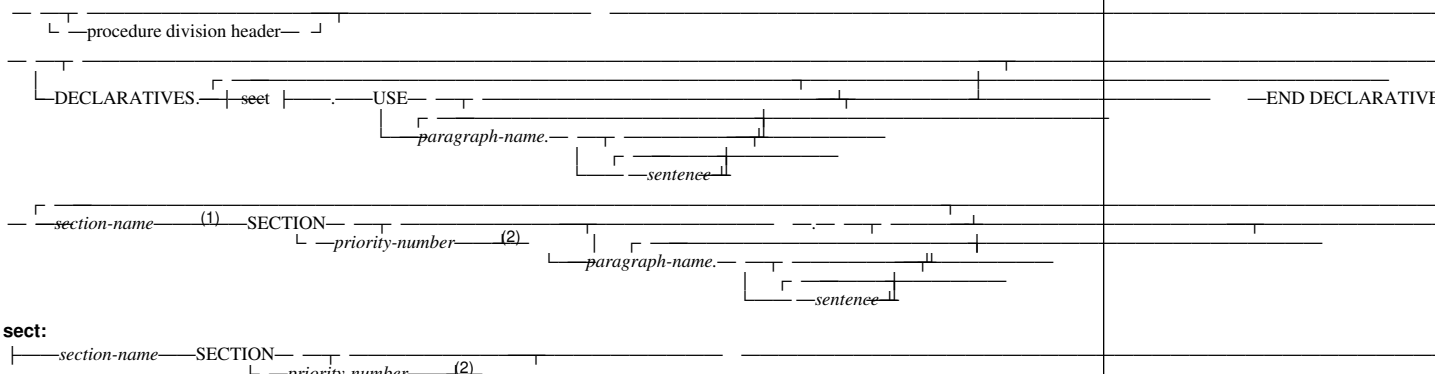
The class Procedure Division contains only method definitions. All methods introduced in a COBOL class compilation unit must be defined in that compilation unit's Procedure Division.

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 226.

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, 1998

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 318.

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 227.

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

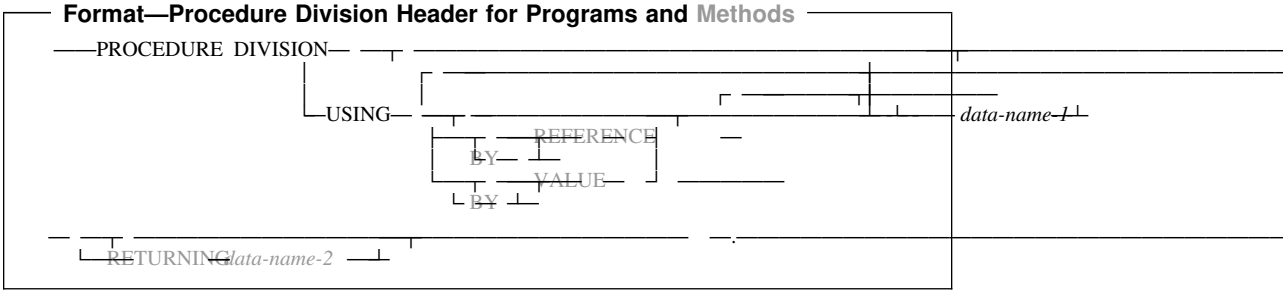
The following special registers are allocated on a per-invocation basis for methods; thus, they are in initial state on each method entry.

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.

Workstation For AIX, OS/2, and Windows, command-line arguments are always passed in as native data types. If you specify the host data type compiler options (CHAR(EBCDIC), FLOAT(HEX), or BINARY(S390)), you must specify the NATIVE

phrase on any arguments with data types affected by these compiler options. (Note, the BINARY compiler option is only applicable to OS/2 and Windows programs.) ◀ Workstation

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 341.

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 either a level 01 or 77 entry in the Linkage Section.

Data-name-2 is an output-only parameter. The initial state of data-name-2 has an undefined and unpredictable value when the program or method is entered. You must initialize data-name-2 in the program or method before you reference its value. When a program or method returns to its invoker, the final 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 287 or "INVOKE Statement" on page 337.

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).

When the RETURNING phrase is specified on the PROCEDURE DIVISION header of a program or method, the CALL or INVOKE statement used to pass control to the program or method must also specify a RETURNING phrase. The data-name-2 and the identifier specified on the CALL or INVOKE RETURNING must have the same PICTURE, USAGE, SIGN, SYNCHRONIZE, JUSTIFIED, and BLANK WHEN ZERO clauses (except that PICTURE clause currency symbols 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.

Declarations

Host Under OS/390 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. Host

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:

- They are operands of the USING phrase of the Procedure Division header or the ENTRY statement
- They are operands of SET ADDRESS OF, CALL...BY REFERENCE ADDRESS 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 533 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.

The USE statement has three formats:

1. EXCEPTION declarative (see “USE Statement” on page 533)
2. DEBUGGING declarative (see “USE Statement” on page 533)
3. LABEL declarative (see “USE Statement” on page 533)

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.

As IBM extensions, the following apply to declarative procedures:

For AIX, OS/2, Windows, OS/390, and VM:

A declarative procedure can be performed from a nondeclarative procedure.

Additionally for OS/390 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 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.

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 there is an eventual 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.

Procedures

Section-header

A **section-name** followed by the key word SECTION, optionally followed, by a **priority-number**, followed by a separator period.

Section-headers are optional after the key words END DECLARATIVES or if there are no declaratives.

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

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.

You cannot specify priority-numbers:

In a method definition
In a program that is declared with the RECURSIVE attribute
In a program that specifies the THREAD compiler option (Workstation 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.

If there are no declaratives (format-2), a paragraph-name is not required in the 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.

As an IBM extension, all paragraphs do not need to be contained within sections, 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.

Arithmetic Expressions

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 .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 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 table:

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

Arithmetic with Date Fields

Arithmetic operations that include a date field are restricted to:

- Adding a non-date to a date field
- Subtracting a non-date from a date field
- Subtracting a date field from a compatible date field

Date field operands are compatible if they have the same date format except for the year part, which may be windowed or expanded.

The following operations are not allowed:

- Any operation between incompatible dates

Adding two date fields

Subtracting a date field from a non-date

Unary minus, applied to a date field

Division, exponentiation, or multiplication of or by a date field

Arithmetic expressions that specify a year-last date field

Arithmetic statements that specify a year-last date field, except as a receiving data item when the sending field is a non-date

The following pages describe the result of using date fields in the supported addition and subtraction operations.

For more information on using date fields in arithmetic operations, see:

"ADD Statement" on page 282
"COMPUTE Statement" on page 300
"SUBTRACT Statement" on page 425

Notes:

1. Arithmetic operations treat date fields as numeric items; they do not recognize any date-specific internal structure. For example, adding 1 to a windowed date field containing the value 991231 (that might be used in an application to represent December 31, 1999) results in the value 991232, not 000101.
2. When used as operands in arithmetic expressions or arithmetic statements, windowed date fields are automatically expanded according to the century window specified by the YEARWINDOW compiler option. Host When the DATEPROC(TRIG) compiler option is in effect, this expansion is sensitive to trigger values in the windowed date field. Host For details of Host both regular and trigger-sensitive Host windowed expansion, see "Semantics of Windowed Date Fields" on page 165.

Addition Involving Date Fields

The following table shows the result of using a date field with a compatible operand in an addition.

Table 17. Results of Using Date Fields in Addition

	Second Operand	
First Operand	**Non-date**	**Date field**
Non-date	Non-date	Date field
Date field	Date field	Not allowed

For details on how a result is stored in a receiving field, see "Storing Arithmetic Results That Involve Date Fields" on page 237.

Subtraction Involving Date Fields

The following table shows the result of using a date field with a compatible operand in the subtraction:

first operand – second operand

In a SUBTRACT statement, these operands appear in the reverse order:

SUBTRACT *second operand* FROM *first operand*

Table 18. Results of Using Date Fields in Subtraction

	Second Operand	
First Operand	**Non-date**	**Date field**
Non-date	Non-date	Not allowed
Date field	Date field	Non-date

Storing Arithmetic Results That Involve Date Fields

The following statements perform arithmetic, then store the result, or sending field, into one or more receiving fields:

ADD
COMPUTE
DIVIDE
MULTIPLY
SUBTRACT

Note: In a MULTIPLY statement, only GIVING identifiers can be date fields. In a DIVIDE statement, only GIVING identifiers or the REMAINDER identifier can be date fields.

Any windowed date fields that are operands of the arithmetic expression or statement are treated as if they were expanded before use, as described under “Semantics of Windowed Date Fields” on page 165.

If the sending field is a date field, then the receiving field must be a compatible date field. That is, both fields must have the same date format, except for the year part, which may be windowed or expanded.

If the ON SIZE ERROR clause is not specified on the statement, the store operation follows the existing COBOL rules for the statement, and proceeds as if the receiving and sending fields (after any automatic expansion of windowed date field operands or result) were both non-dates.

When the ON SIZE ERROR clause is specified, Table 19 on page 239 shows how these statements store the value of a sending field in a receiving field, where either field may be a date field.

Table 19 on page 239 uses the following terms to describe how the store is performed:

Arithmetic Expressions

Non-windowed

The statement performs the store with no special date-sensitive size error processing, as described under “SIZE ERROR Phrases” on page 266.

Windowed...

...with non-date sending field

The non-date sending field is treated as a windowed date field compatible with the windowed date receiving field, but with the year part representing the number of years since 1900. (This representation is similar to a windowed date field with a base year of 1900, except that the year part is not limited to a positive number of at most 2 digits.) The store proceeds as if this assumed year part of the sending field were expanded by adding 1900 to it.

...with date sending field

The store proceeds as if all windowed date field operands had been expanded as necessary, so that the sending field is a compatible expanded date field.

Size error processing: For both kinds of sending field, if the assumed or actual year part of the sending field falls within the century window, then the sending field is stored in the receiving field after removing the century component of the year part. That is, the low-order or rightmost 2 digits of the expanded year part are retained, and the high-order or leftmost 2 digits are discarded.

If the year part does not fall within the century window, then the receiving field is unmodified, and the size error imperative statement is executed when any remaining arithmetic operations are complete.

For example:

```
77   DUE-DATE PICTURE 9(5) DATE FORMAT YYXXX.
77   IN-DATE  PICTURE 9(8) DATE FORMAT YYYYXXX VALUE 1995 1.
     :
     COMPUTE DUE-DATE = IN-DATE + 1
        ON SIZE ERROR imperative-statement
     END-COMPUTE
```

The sending field is an expanded date field representing January 1, 2005. Assuming that 2005 falls within the century window, the value stored in DUE-DATE is 05001—the sending value of 2005001 without the century component 20.

Host

Size error processing and trigger values: If the DATEPROC(TRIG) compiler option is in effect, and the sending field contains a trigger value (either zero or all nines) the size error imperative statement is executed, and the result is not stored in the receiving field.

A non-date is considered to have a trigger value of all nines if it has a nine in every digit position of its assumed date format. Thus, for a receiving date format of YYXXX,

the non-date value 99,999 is a trigger, but the values 9,999 and 999,999 are not, although the larger value of 999,999 will cause a size error anyway. Host

Table 19. Storing Arithmetic Results Involving Date Fields When ON SIZE ERROR is Specified

	Sending Field	
Receiving Field	**Non-date**	**Date field**
Non-date	Non-windowed	Not allowed
Windowed date field	Windowed	Windowed
Expanded date field	Non-windowed	Non-windowed

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.

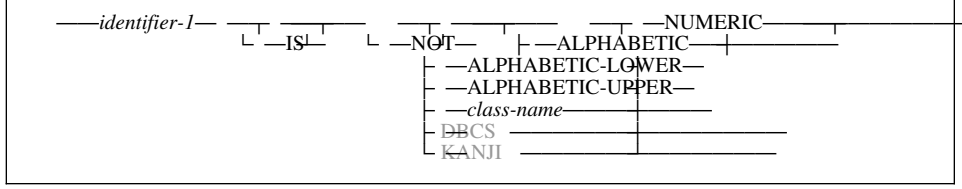
Class Condition

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.

The class condition determines whether the contents of a data item are DBCS or KANJI.

Conditional Expressions

Format



identifier-1
Must reference a data item whose usage is explicitly or implicitly DISPLAY.

Identifier-1 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 DISPLAY or (as IBM extensions) USAGE COMPUTATIONAL-3, or USAGE 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.

DBCS

Host Under OS/390 and VM, the identifier consists entirely of DBCS characters. For DBCS data items, the identifier being tested must be described explicitly or implicitly as USAGE DISPLAY-1. Each byte of the DBCS identifier being tested can contain characters that range in value from X'00' through X'FF'. Host

Workstation Under AIX, OS/2, and Windows, the identifier contains DBCS characters that correspond to valid OS/390 DBCS characters. Workstation

For all platforms, a range check is performed on the data portion of the item for valid character representation. The valid range is X'41' through X'FE' for both bytes of each DBCS character and X'4040' for the DBCS blank. (These ranges are for the equivalent DBCS character representation for OS/390, not the actual DBCS character value ranges of the workstation DBCS characters.)

KANJI

Host Under OS/390 and VM, the identifier consists entirely of DBCS characters. For KANJI data items, the identifier being tested must be described explicitly or implicitly as USAGE DISPLAY-1. Each byte of the DBCS identifier being tested can contain characters that range in value from X'00' through X'FF'. Host

Workstation Under AIX, OS/2, and Windows the identifier contains DBCS characters that correspond to valid OS/390 DBCS characters. Workstation

For all platforms, a range check is performed on the data portion of the item for valid character representation. The valid range is from X'41' through X'7F' for the first byte, from X'41' through X'FE' for the second byte, and X'4040' for the DBCS blank. (These ranges are for the equivalent DBCS character representation for OS/390, not the actual DBCS character value ranges of the workstation DBCS characters.)

The class test is not valid for items defined as USAGE IS INDEX, as these items do not belong to any class or category.

The class test is not valid for items defined as USAGE IS POINTER or USAGE IS PROCEDURE-POINTER, as these items do not belong to any class or category.

The class condition cannot be used for external floating-point (USAGE DISPLAY) or internal floating-point (USAGE COMP-1 and USAGE COMP-2) items.

Conditional Expressions

Table 20 shows valid forms of the class test.

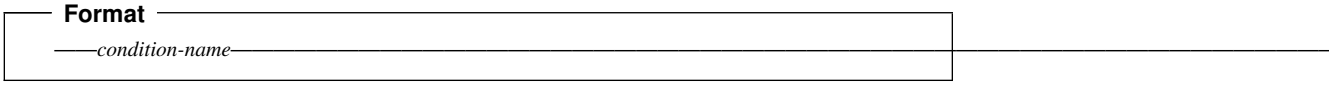
Table 20. Valid Forms of the Class Test for Different Types of Identifiers

Type of Identifier	Valid Forms of the Class Test	
Alphabetic	ALPHABETIC ALPHABETIC-LOWER ALPHABETIC-UPPER class-name	NOT ALPHABETIC NOT ALPHABETIC-LOWER NOT ALPHABETIC-UPPER NOT class-name
Alphanumeric, Alphanumeric-edited, or Numeric-edited	ALPHABETIC ALPHABETIC-LOWER ALPHABETIC-UPPER NUMERIC class-name	NOT ALPHABETIC NOT ALPHABETIC-LOWER NOT ALPHABETIC-UPPER NOT NUMERIC NOT class-name
External-Decimal or Internal-Decimal	NUMERIC	NOT NUMERIC
DBCS	DBCS KANJI	NOT DBCS 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.

Condition-names with DBCS and floating-point values are allowed.

The following example illustrates the use of conditional variables and condition-names:

```
 1  AGE-GROUP              PIC  99.
    88  INFANT             VALUE  .
    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  )
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.

Condition-Name Conditions and Windowed Date Field Comparisons

If the conditional variable is a windowed date field, then the values associated with its condition-names are treated like values of the windowed date field; that is, they are treated as if they were converted to expanded date format, as described under "Semantics of Windowed Date Fields" on page 165.

For example, given YEARWINDOW(1945), specifying a century window of 1945–2044, and the following definition:

```
5   DATE-FIELD     PIC 9(6) DATE FORMAT YYXXXX.
    88   DATE-TARGET          VALUE   5122 .
```

then a value of 051220 in DATE-FIELD would cause the following condition to be true:

```
IF DATE-TARGET...
```

because the value associated with DATE-TARGET and the value of DATE-FIELD would both be treated as if they were prefixed by "20" before comparison.

However, the following condition would be false:

```
IF DATE-FIELD =  5122 ...
```

because, in a comparison with a windowed date field, literals are treated as if they are prefixed by "19", regardless of the century window. So the above condition effectively becomes:

```
IF 2  5122  = 19 5122 ...
```

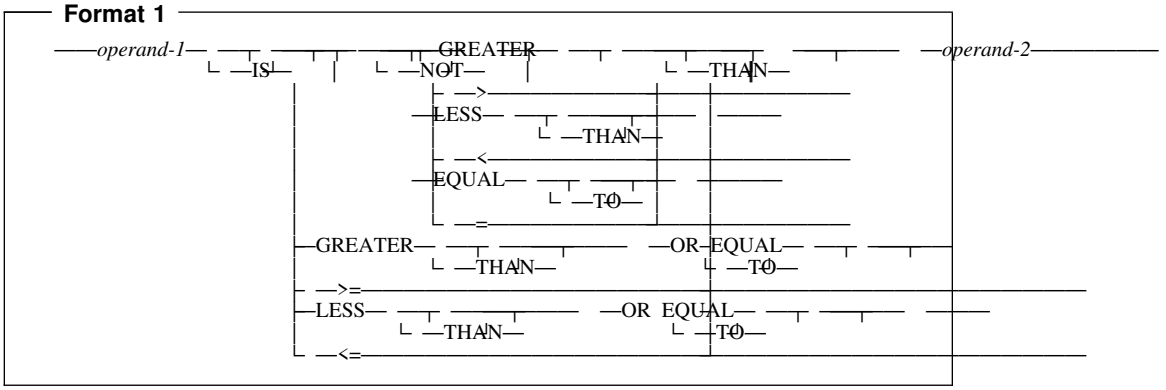
For more information on using windowed date fields in conditional expressions, see "Date Fields" on page 244.

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.

Conditional Expressions

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.

The relation condition must contain at least one reference to an identifier.

The relational operator specifies the type of comparison to be made. Table 21 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 21. 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

Date Fields

Date fields can be alphanumeric, external decimal, or internal decimal; the existing rules for the validity and mode (numeric or nonnumeric) of comparing such items still

apply. For example, an alphanumeric date field cannot be compared with an internal decimal date field. In addition to these rules, two date fields can be compared only if they are compatible; they must have the same date format except for the year part, which may be windowed or expanded.

For year-last date fields, the only comparisons that are supported are IS EQUAL TO and IS NOT EQUAL TO between two year-last date fields with identical date formats, or between a year-last date field and a non-date.

Table 22 on page 246 shows supported comparisons for non-year-last date fields. This table uses the following terms to describe how the comparisons are performed:

Non-windowed
The comparison is performed with no windowing, as if the operands were both non-dates.

Windowed
The comparison is performed as if:

1. Any windowed date field in the relation were expanded according to the century window specified by the YEARWINDOW compiler option, as described under "Semantics of Windowed Date Fields" on page 165.

   Host This expansion is sensitive to trigger values in the date field comparand if the DATEPROC(TRIG) compiler option is in effect. Host

2. Any repetitive alphanumeric figurative constant were expanded to the size of the windowed date field with which it is compared, giving an alphanumeric non-date comparand. Repetitive alphanumeric figurative constants include ZERO (in an alphanumeric context), SPACE, LOW-VALUE, HIGH-VALUE, QUOTE and ALL literal.

3. Any non-date operands were treated as if they had the same date format as the date field, but with a base year of 1900.

   Host If the DATEPROC(NOTRIG) compiler option is in effect, the comparison is performed as if the non-date operand were expanded by assuming 19 for the century part of the expanded year.

   If the DATEPROC(TRIG) compiler option is in effect, the comparison is sensitive to date trigger values in the non-date operand. For alphanumeric operands, these trigger values are LOW-VALUE, HIGH-VALUE, and SPACE. For alphanumeric and numeric operands compared with windowed date fields with at least one X in the DATE FORMAT clause (that is, windowed date fields other than just a windowed year), values of all zeros or all nines are also treated as triggers. If a non-date operand contains a trigger value, the comparison proceeds as if the non-date operand were expanded by copying the trigger value to the assumed century part of the expanded year. If the non-date operand does not contain a trigger value, the century part of the expanded year is assumed to be 19. Host

The comparison is then performed according to normal COBOL rules. Nonnumeric comparisons are not changed to numeric comparisons by the prefixing of the century value.

Table 22. Comparisons with Date Fields

	Second Operand		
First Operand	**Non-date**	**Windowed date field**	**Expanded date field**
Non-date	Non-windowed	Windowed[1]	Non-windowed
Windowed date field	Windowed[1]	Windowed	Windowed
Expanded date field	Non-windowed	Windowed	Non-windowed

Note:

1. When compared with windowed date fields, non-dates are assumed to contain a windowed year relative to 1900. For details, see items 3 and 4 under the definition of "Windowed" comparison.

Relation conditions can contain arithmetic expressions. For information about the treatment of date fields in arithmetic expressions, see "Arithmetic with Date Fields" on page 235.

DBCS Items

Host Under OS/390 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. Host

Workstation Under AIX, OS/2, and Windows, comparisons of DBCS data items and literals are based on a collation sequence according to the COLLSEQ compiler option:

If the COLLSEQ(NATIVE) compiler option is in effect, then the collating sequence is determined by the locale. For information on the locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 568.

Otherwise, the collating sequence is determined by the binary values of the DBCS characters. Workstation

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.

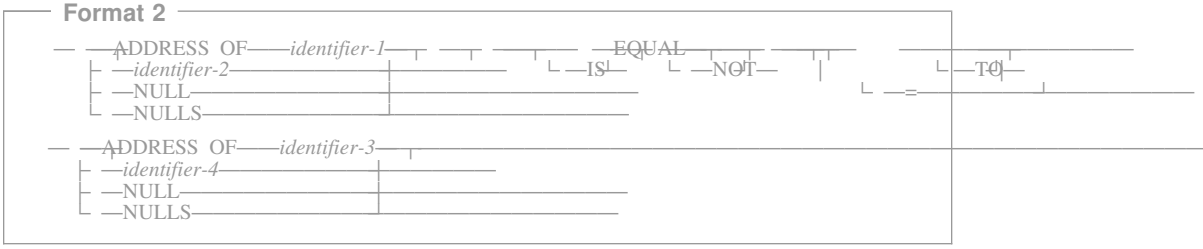
Pointer Data Items

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 23 summarizes the permissible comparisons for USAGE IS POINTER, NULL, and ADDRESS OF.

Table 23. 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

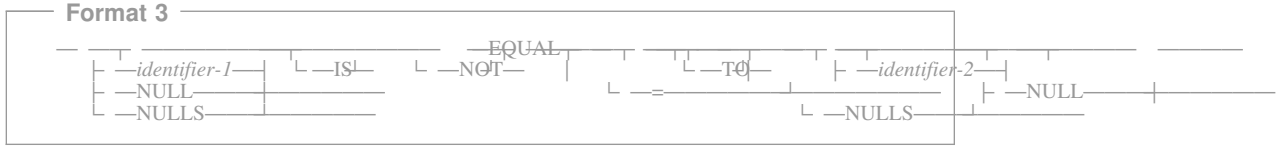
Only EQUAL and NOT EQUAL are allowed as relational operators when specifying procedure-pointer data items. Procedure-pointer data items are items defined explicitly as USAGE IS PROCEDURE-POINTER.

Conditional Expressions

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 procedure-pointer data items.

Format 3



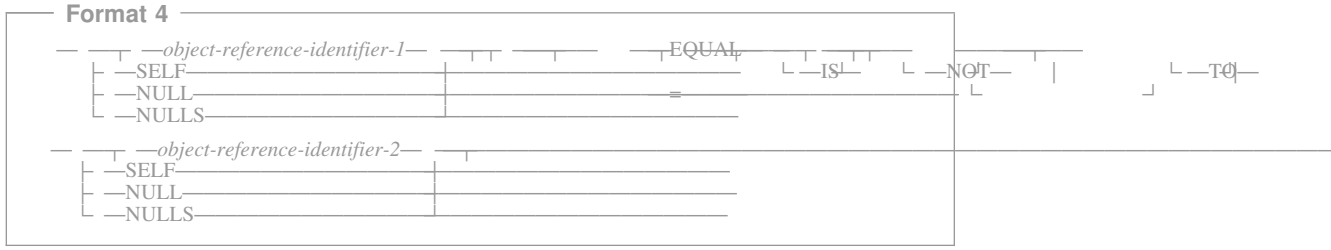
identifier-1
identifier-2
Must be described as USAGE IS PROCEDURE-POINTER.

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 24 on page 249 summarizes permissible comparisons with **numeric** operands.

The symbols used in Table 24 and Table 25 are as follows:

NN = Comparison for nonnumeric operands
NU = Comparison for numeric operands
Blank = Comparison is not allowed.

Table 24. Permissible Comparisons with Numeric Second Operands

	Second Operand								
First Operand	**ZR**	**NL**	**ED**	**BI**	**AE**	**ID**	**IFP**	**EFP**	**FPL**
Nonnumeric Operand									
Group (GR)	NN	NN[1]	NN[1]					NN	
Alphabetic (AL)	NN	NN[1]	NN[1]					NN	
Alphanumeric (AN)	NN	NN[1]	NN[1]					NN	
Alphanumeric-edited (ANE)	NN	NN[1]	NN[1]					NN	
Numeric-Edited (NE)	NN	NN[1]	NN[1]					NN	
Figurative Constant (FC[2])			NN[1]					NN	
Nonnumeric Literal (NNL)			NN[1]					NN	
Numeric Operand									
Figurative Constant ZERO (ZR)			NU	NU	NU	NU	NU	NU	
Numeric Literal (NL)			NU	NU	NU	NU	NU	NU	
External Decimal (ED)	NU	NU	NU	NU	NU	NU	NU	NU	NU
Binary (BI)	NU	NU	NU	NU	NU	NU	NU	NU	NU
Arithmetic Expression (AE)	NU	NU	NU	NU	NU	NU	NU	NU	NU
Internal Decimal (ID)	NU	NU	NU	NU	NU	NU	NU	NU	NU
Internal Floating-point (IFP)	NU	NU	NU	NU	NU	NU	NU	NU	NU
External Floating-Point (EFP)	NU	NU	NU	NU	NU	NU	NU	NU	NU
Floating-point Literal (FPL)			NU	NU	NU	NU	NU	NU	

Note:

1 Integer item only.
2 Includes all figurative constants except ZERO.

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 548.)

Conditional Expressions

Workstation 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 568.
Workstation

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 25 summarizes permissible comparisons with **nonnumeric** operands.

Table 25. 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.

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 352.

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.

In the comparison of an index-name with an arithmetic expression, the occurrence number that corresponds to the value of the index-name is compared with the arithmetic expression.

Since an integer function can be used wherever an arithmetic expression can be used, this extension allows you to compare an index-name to an integer or numeric 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 26 on page 253 shows valid comparisons for index-names and index data items.

Table 26. 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.

Host Under OS/390 and VM, the comparison is based on a binary collating sequence of the hexadecimal values of the DBCS characters. Host

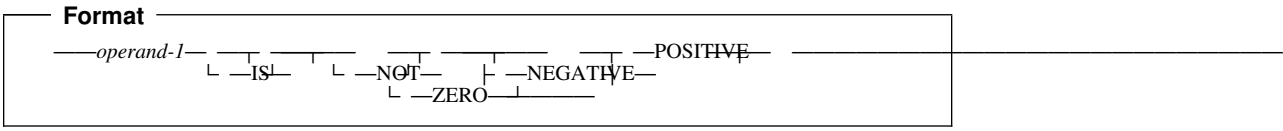
Workstation Under AIX, OS/2, and Windows if the collating sequence specified is NATIVE (explicitly or by default), the comparisons of characters are based on the collating sequence indicated by the locale setting. For more information on locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 568. Workstation

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-1

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. Operand-1 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.

Host Under OS/390 and VM, if you are using the NUMPROC compiler option, the results of the sign condition test can be affected. For details, see the *IBM COBOL for OS/390 & VM Programming Guide*. Host

Date Fields in Sign Conditions

The operand in a sign condition can be a date field, but is treated as a non-date for the sign condition test. Thus, if the operand is an identifier of a windowed date field, date windowing is not done, so the sign condition may be used to test a windowed date field for an all-zero value.

However, if the operand is an arithmetic expression, then any windowed date fields in the expression will be expanded during the computation of the arithmetic result, prior to using the result for the sign condition test.

For example, given that:

Identifier WIN-DATE is defined as a windowed date field, and contains a value of zero

Compiler option DATEPROC is in effect

Compiler option YEARWINDOW(*starting-year*) is in effect, with a *starting-year* other than 1900

then this sign condition would evaluate to true:

```
WIN-DATE IS ZERO
```

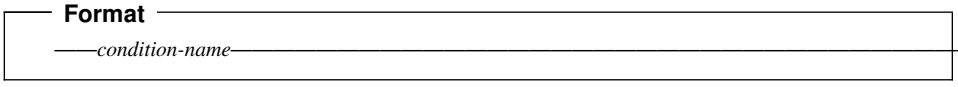
whereas this sign condition would evaluate to false:

```
WIN-DATE +  IS ZERO
```

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 89.)

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 the *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 table shows the logical operators and their meanings.

Table 27. Logical Operators and Their Meanings

Logical Operator	Name	Meaning
AND	Logical conjunction	The truth value is **true** when both conditions are true.
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:

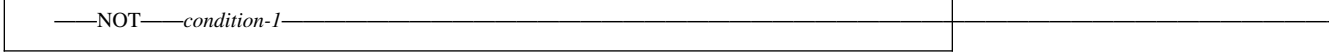
A negated simple condition
A combined condition (which can be negated)

Conditional Expressions

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:

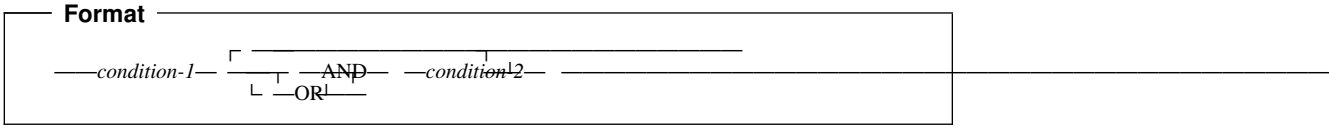
```
NOT A IS EQUAL TO B.

NOT (A IS EQUAL TO B).
```

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 28. 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 29 illustrates the relationships between logical operators and conditions C1 and C2.

Table 29. Logical Operators and Evaluation Results of Combined Conditions

Value for C1	Value for C2	C1 AND C2	C1 OR C2	NOT (C1 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:

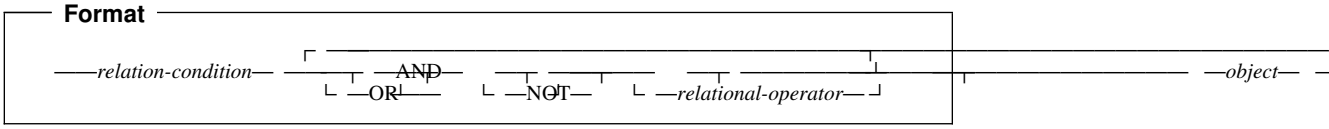
1. (NOT (A IS GREATER THAN B)) is evaluated, giving some intermediate truth value, **t1**. If **t1** is true, the combined condition is true, and no further evaluation takes place. If **t1** is false, evaluation continues as follows.
2. (A + B) is evaluated, giving some intermediate result, **x**.
3. (**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 28 on page 257.

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 table summarizes the rules for forming an abbreviated combined relation condition.

Table 30. 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 31. 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 263). 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 263). Table 32 lists COBOL imperative statements.

Table 32 (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
	EXIT METHOD
	GOBACK
Input-Output	
	ACCEPT identifier
	CLOSE
	DELETE[3]
	DISPLAY
	OPEN
	READ[4]
	REWRITE[3]
	START[3]
	STOP literal
	WRITE[5]

Statement Categories

Table 32 (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 239.) Table 33 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 33 (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 33 (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
Program or Method Linkage	
	CALL...ON OVERFLOW
	CALL...ON EXCEPTION
	CALL...NOT ON EXCEPTION
	INVOKE...ON EXCEPTION
	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 Operations

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
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 512.

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, 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:

```
5  ITEM-1 OCCURS 6.
 1   ITEM-A  PIC  S9(3).
 1   ITEM-B  PIC  +99.9.
 1   ITEM-C  PIC  X(4).
 1   ITEM-D  REDEFINES  ITEM-C  PIC  9(4).
 1   ITEM-E  USAGE  COMP-1.
 1   ITEM-F  USAGE  INDEX.
5  ITEM-2.
 1   ITEM-A  PIC  99.
 1   ITEM-B  PIC  +9V9.
 1   ITEM-C  PIC  A(4).
 1   ITEM-D  PIC  9(4).
 1   ITEM-E  PIC  9(9)  USAGE  COMP.
 1   ITEM-F  USAGE  INDEX.
```

Statement Operations

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.

In a floating-point arithmetic operation, the ROUNDED phrase has no effect; the result of a floating-point operation is **always** rounded. For more information on floating-point arithmetic expressions, see *IBM COBOL Programming Guide* for your platform.

SIZE ERROR Phrases

A size error condition can occur in four 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

When the result of an arithmetic statement is stored in a windowed date field, and the year of the result falls outside the century window. For example, given YEARWINDOW(1940), which specifies a century window of 1940–2039, the following SUBTRACT statement causes a size error:

```
1  WINDOWED-YEAR  DATE FORMAT YY PICTURE 99
                  VALUE IS 5 .
   ⋮

   SUBTRACT 2 FROM WINDOWED-YEAR
            ON SIZE ERROR imperative-statement
```

The size error occurs because the result of the subtraction, a windowed date field, has an effective year value of 1930, which falls outside the century window. For details on how windowed date fields are treated as if they were converted to expanded date format, see "Subtraction Involving Date Fields" on page 237.

For more information on how size errors can occur when using date fields, see "Storing Arithmetic Results That Involve Date Fields" on page 237.

In an exponential expression, as indicated in the following table:

Table 34. 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 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.

Statement Operations

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).
```

The composite of operands can be more than 18 digits. For more information, see the section on intermediate results in the *IBM COBOL Programming Guide* for your platform.

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 35. How the Composite of Operands is Determined

Statement	Determination of the Composite of Operands
SUBTRACT 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 45 on page 363 and Table 46 on page 364.

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 36 on page 271.

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 (OS/390 and VM only) or to line sequential files (Workstation only). For more information on data-name-8, see "FILE STATUS Clause" on page 122.

Table 36 (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. 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.
		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 OS/390 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.

Statement Operations

Table 36 (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. 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: 1. The EXTEND or OUTPUT phrase was specified but the file would not support write operations. 2. The I-O phrase was specified but the file would not support the input and output operations permitted. 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. Under AIX, OS/2, and Windows, file status 39 is not supported for line sequential files or Btrieve files.

Table 36 (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. 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: 1. The preceding READ statement was unsuccessful but did not cause an at end condition 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 36 (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 OS/390 and VM: Password failure. Under AIX, OS/2, and Windows: Authorization failure.
		2	Logic error.
		3	For all files, except QSAM: Resource not available.
		4	For VSAM under OS/390 and VM with CMPR2 compiler-option only: No file position indicator for sequential request. 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 OS/390 and VM: An OPEN statement with the OUTPUT or EXTEND phrase was attempted for an optional file, but no DD statement was specified for the file. For QSAM file under OS/390 and VM: An OPEN statement with the OUTPUT or EXTEND phrase was attempted for an optional file, but no DD statement was specified for the file and the CBLQDA(OFF) run-time option was specified. Under AIX, OS/2, and Windows: File system not available.
		7	For VSAM only under OS/390 and VM: OPEN statement execution successful: File integrity verified. 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 85.) 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 36 on page 271.)
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.

Both the INVALID KEY phrase and the EXCEPTION/ERROR procedure can be omitted.

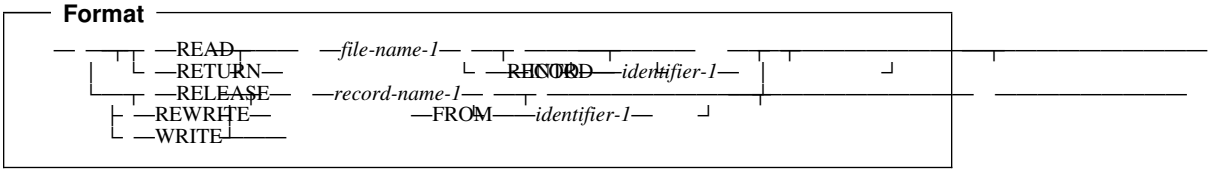
If the invalid key condition does not exist after execution of the input-output operation, the INVALID KEY phrase is ignored, if specified, and the following actions are taken:

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.

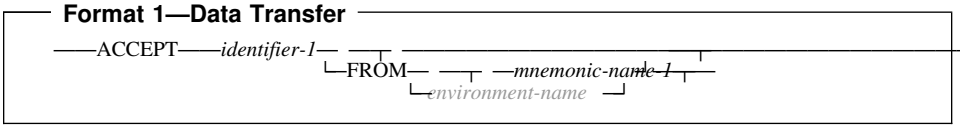
Procedure Division Statements

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



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.

Workstation 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). You can create a byte stream file in your COBOL program using line sequential file I-O or with the DISPLAY statement. (Most text editors can be used to create a byte stream file as well.)

The input file **cannot** be a VSAM, Btrieve, SFS, 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. Workstation

identifier-1

Can be any group item, or an elementary alphabetic, alphanumeric, alphanumeric-edited, numeric-edited or external decimal item.

It can also be a DBCS data item or an external floating-point item.

© Copyright IBM Corp. 1991, 1998

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 89.

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.

Host Under OS/390 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. Host

Console

Host Under OS/390 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. Host

Workstation 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.
Workstation

environment-name

A valid **environment-name** can be specified. See Table 8 on page 91 for a list of valid environment-names.

Workstation 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.
Workstation

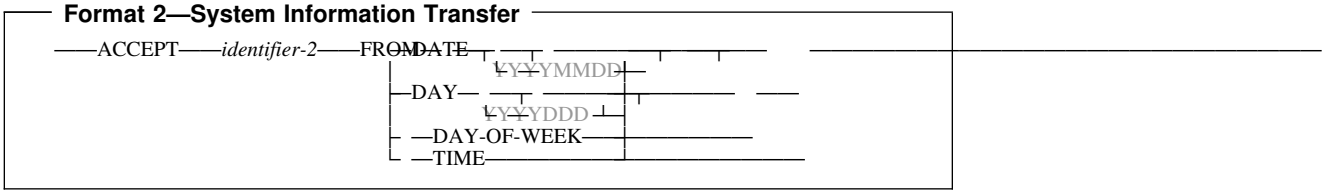
Note: If the device is the same as that used for READ statements, results are unpredictable.

System Information Transfer

System information contained in the specified conceptual data items DATE, DATE YYYYMMDD, DAY, DAY YYYYDDD, 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 352.

ACCEPT Statement

Format 2—System Information Transfer



identifier-2
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 or the time of day as carried by the system, which can be useful in identifying when a particular run of an object program was executed. You can also use Format 2 to supply the date in headings and footings.

Note: The current date and time is also accessible via the date/time intrinsic function CURRENT-DATE, which also supports 4-digit year values and provide additional information (see "Intrinsic Functions" on page 447).

DATE, DATE YYYYMMDD, DAY, DAY YYYYDDD, DAY-OF-WEEK, and TIME

The conceptual data items DATE, DATE YYYYMMDD, DAY, DAY YYYYDDD, DAY-OF-WEEK, and TIME implicitly have USAGE DISPLAY. Because these are conceptual data items, they cannot be described in the COBOL program.

DATE
Has the implicit PICTURE 9(6). If the DATEPROC compiler option is in effect, then the returned value has implicit DATE FORMAT YYXXXX, and identifier-2 must be defined with this date format.

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: 95 427

DATE YYYYMMDD
Has the implicit PICTURE 9(8). If the DATEPROC compiler option is in effect, then the returned value has implicit DATE FORMAT YYYYXXXX, and identifier-2 must be defined with this date format.

The sequence of data elements (from left to right) is:

4 digits for the year
2 digits for the month
2 digits for the day

Thus, 27 April 1995 is expressed as: 1995 427

DAY

Has the implicit PICTURE 9(5). If the DATEPROC compiler option is in effect, then the returned value has implicit DATE FORMAT YYXXX, and identifier-2 must be defined with this date format.

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 YYYYDDD

Has the implicit PICTURE 9(7). If the DATEPROC compiler option is in effect, then the returned value has implicit DATE FORMAT YYYYXXX, and identifier-2 must be defined with this date format.

The sequence of data elements (from left to right) is:

4 digits for the year
3 digits for the day

Thus, 27 April 1995 is expressed as: 1995117

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
2 represents Tuesday
3 represents Wednesday
4 represents Thursday
5 represents Friday
6 represents Saturday
7 represents Sunday

Thus, Wednesday is expressed as: 3

TIME

Has the implicit PICTURE 9(8).

The sequence of data elements (from left to right) is:

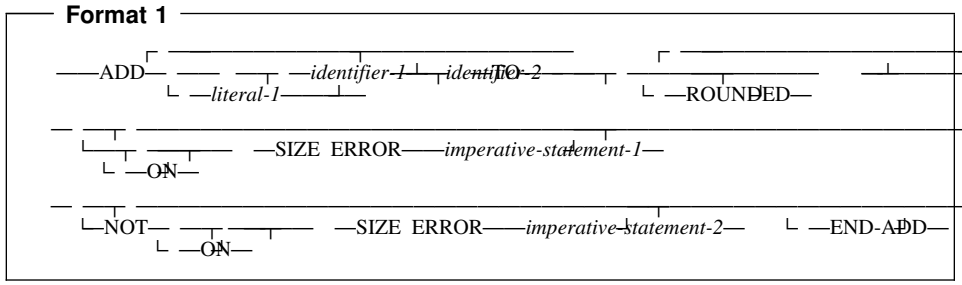
2 digits for hour of day
2 digits for minute of hour
2 digits for second of minute
2 digits for hundredths of second

Thus, 2:41 PM is expressed as: 1441

ADD Statement

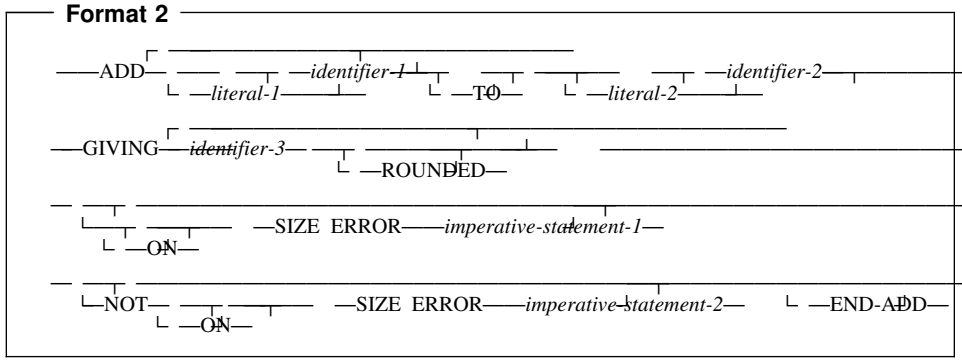
The ADD statement sums two or more numeric operands and stores the result.

Format 1



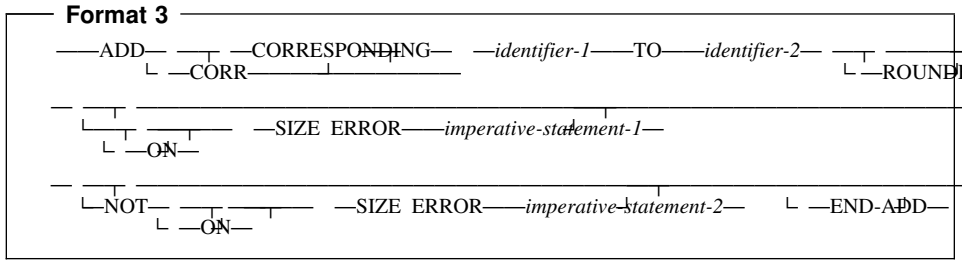
All identifiers or literals preceding the key word TO are added together, and this sum is added to and stored in identifier-2. This process is repeated for each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

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.

The following restrictions apply to date fields:

In Format 1, identifier-2 may specify one or more date fields; identifier-1 must not specify a date field.

In Format 2, either identifier-1 or identifier-2 (but not both) may specify at most one date field. If identifier-1 or identifier-2 specifies a date field, then every instance of identifier-3 must specify a date field that is compatible with the date field specified by identifier-1 or identifier-2. That is, they must have the same date format, except for the year part, which may be windowed or expanded.

If neither identifier-1 nor identifier-2 specifies a date field, then identifier-3 may specify one or more date fields without any restriction on the date formats.

In Format 3, only corresponding elementary items within identifier-2 may be date fields. There is no restriction on the format of these date fields.

A year-last date field is allowed in an ADD statement only as identifier-1 and when the result of the addition is a non-date

There are two steps to determining the result of an ADD statement that involves one or more date fields:

1. Addition: determine the result of the addition operation, as described under "Addition Involving Date Fields" on page 236.
2. Storage: determine how the result is stored in the receiving field. (In Formats 1 and 3, the receiving field is identifier-2; in Format 3, the receiving field is the GIVING identifier-3.) For details, see "Storing Arithmetic Results That Involve Date Fields" on page 237.

literal

Must be a numeric literal.

Floating-point data items and literals can be used anywhere a numeric data item or 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.

The composite of operands can be more than 18 digits. For information on arithmetic intermediate results, see the *IBM COBOL Programming Guide* for your platform.

ADD Statement

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 266.

SIZE ERROR Phrases

For Formats 1, 2, and 3, see “SIZE ERROR Phrases” on page 266.

CORRESPONDING Phrase (Format 3)

See “CORRESPONDING Phrase” on page 265.

END-ADD Phrase

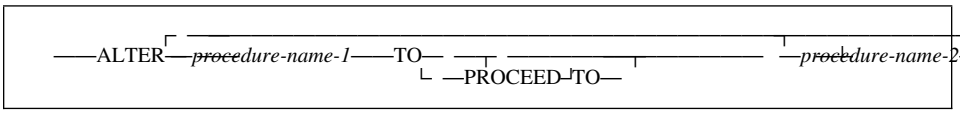
This explicit scope terminator serves to delimit the scope of the ADD statement. END-ADD permits a conditional ADD statement to be nested in another conditional statement. END-ADD can also be used with an imperative ADD statement.

For more information, see “Delimited Scope Statements” on page 263.

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.

Do not use the ALTER statement in programs that have the RECURSIVE attribute, in 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

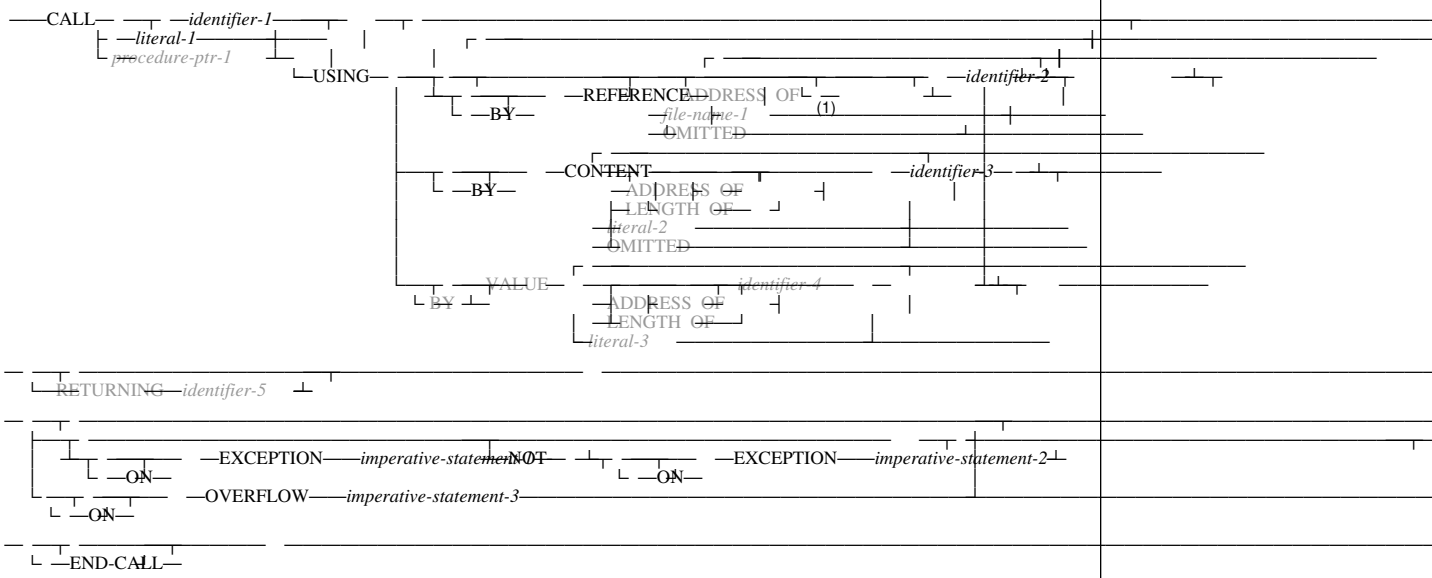
The CALL statement transfers control from one object program to another within the run unit.

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.

Programs defined with the RECURSIVE attribute can execute a CALL statement that directly or indirectly CALLs itself.

Do not specify the name of a class or a method in the CALL statement.

Format



Note:

1 *File-name-1* is supported on OS/390 and VM only.

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 77 and also the description of the PGMNAME compiler option in the *IBM COBOL Programming Guide* for your platform.

Identifier-1 can be an alphabetic or zoned decimal data item. Identifier-1 cannot be a windowed date field.

CALL Statement

procedure-pointer-1

Must be defined with USAGE IS PROCEDURE-POINTER, and must be set to a valid program entry point; otherwise, the results of the CALL statement are undefined.

After a program has been canceled by COBOL, released by PL/I or C, or deleted by assembler, any procedure-pointers that had been set to that program's entry 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.

When the called subprogram is entered through an ENTRY statement, literal-1 or the contents of identifier-1 must be the same as the name specified in the called subprogram'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 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 227.

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.

The sequence of appearance of the identifiers in the USING phrase of the CALL statement and in the corresponding USING phrase in the called program's ENTRY statement determines the correspondence between the identifiers used by the calling and 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.

The BY CONTENT, BY REFERENCE and BY VALUE phrases apply to parameters that 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 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
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.

file-name-1 (OS/390 and VM Only)
Host Under OS/390 and VM, a file-name for a QSAM file. See *IBM COBOL for OS/390 & VM Programming Guide* for details on using file-name with the CALL statement. Host

ADDRESS OF Special Register
For information on the ADDRESS OF special register, see “ADDRESS OF” on page 10.

OMITTED
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
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.

literal-2
Can be:

- A nonnumeric literal
- A figurative constant (except ALL literal or NULL/NULLS)
- A DBCS literal

LENGTH OF Special Register
For information on the LENGTH OF special register, see “LENGTH OF” on page 12.

ADDRESS OF Special Register
For information on the ADDRESS OF special register, see “ADDRESS OF” on page 10.

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 10.

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 12.

literal-3
Must be one of the following:

Numeric literal
ZERO
1-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 PICTURE clause currency symbols 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.

Note: The CALL... RETURNING data item is an output-only parameter. On entry to the called program, the initial state of the PROCEDURE DIVISION RETURNING data item has an undefined and unpredictable value. You must initialize the PROCEDURE DIVISION RETURNING data item in the called program before you reference its value. The value that is passed back to the calling program is the final value of the PROCEDURE DIVISION RETURNING data item when the called program returns.

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

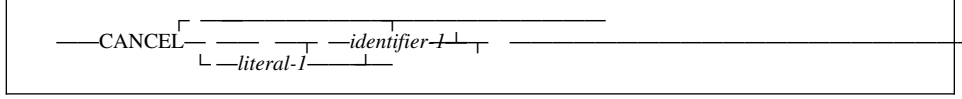
This explicit scope terminator serves to delimit the scope of the CALL statement. END-CALL permits a conditional CALL statement to be nested in another conditional statement. END-CALL can also be used with an imperative CALL statement.

For more information, see “Delimited Scope Statements” on page 263.

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 77 and also the description of the PGMNAME compiler option in the *IBM COBOL Programming Guide* for your platform.

Identifier-1 can be alphabetic or zoned decimal data item. It cannot be a windowed date field.

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.

The program-name referenced in the CANCEL statement can be affected by the PGMNAME compiler option. For details, see the *IBM COBOL Programming Guide* for your platform.

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 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 OS/390 and VM, has not been dynamically called in this run unit by another COBOL for MVS® & VM, COBOL for OS/390 & 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 called and has not yet executed an EXIT PROGRAM or a GOBACK statement.

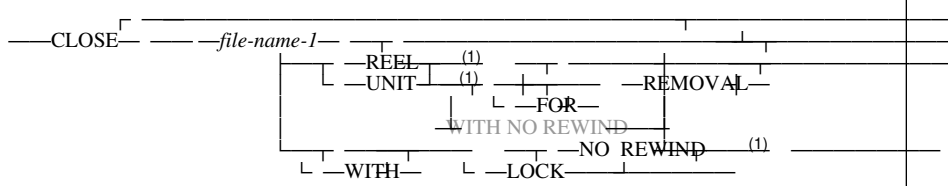
A program can, however, cancel a program that it did not call, providing that, in the calling hierarchy, it is higher than or equal to the program it is canceling. For example:

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.

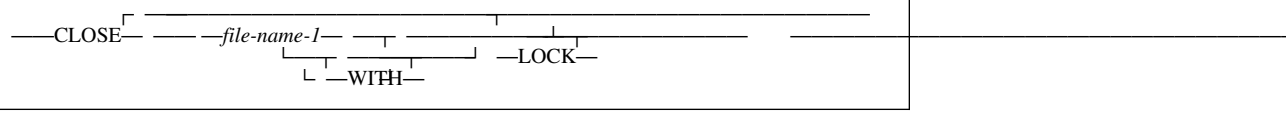
Format 1—Sequential



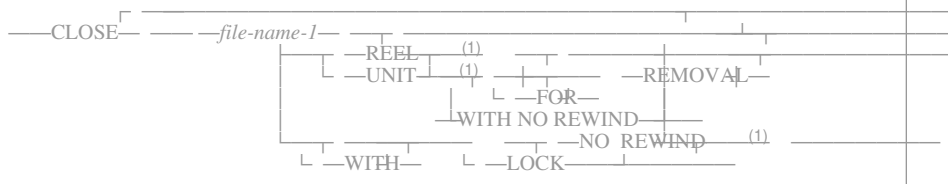
Note:

1 Under OS/390, 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 OS/390 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.

WITH NO REWIND and FOR REMOVAL

Under OS/390 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 37 on page 298 for sequential files
Table 38 on page 298 for indexed and relative files
Table 39 on page 298 for line sequential files

CLOSE Statement

The meaning of each key letter is shown in Table 40 on page 298.

Table 37. 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 38. Indexed and Relative File Types and CLOSE Statement Phrases

CLOSE Statement Phrases	Action
CLOSE	C
CLOSE WITH LOCK	C,E

Table 39. Line Sequential File Types and CLOSE Statement Phrases

CLOSE Statement Phrases	Action
CLOSE	C
CLOSE WITH LOCK	C,E

Table 40 (Page 1 of 2). Meanings of Key Letters for Sequential File Types

Key	Actions Taken
A	**Previous Volumes Unaffected** **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. **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 40 (Page 2 of 2). Meanings of Key Letters for Sequential File Types

Key	Actions Taken
C	**Close File** **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. 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. If the file is not at its end, standard system closing procedures are performed, but there is no ending label processing. **Output Files**—If label records are specified, standard ending label procedures are performed. Standard system closing procedures are then performed. 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** **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. **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. **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.

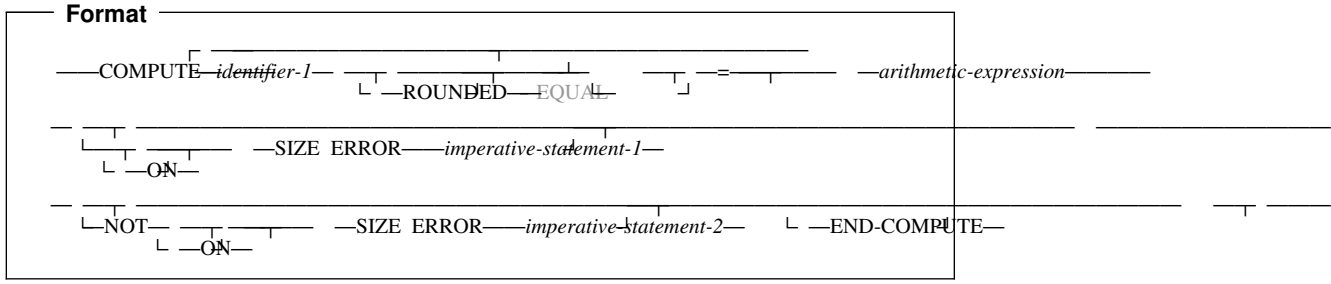
COMPUTE Statement

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



identifier-1

Must name elementary numeric item(s) or elementary numeric-edited item(s).

Can name an elementary floating-point data item.

The word EQUAL can be used in place of =.

If identifier-1 or the result of the arithmetic expression (or both) are date fields, see "Storing Arithmetic Results That Involve Date Fields" on page 237 for details on how the result is stored in identifier-1. If a year-last date field is specified as identifier-1, then the result of the arithmetic expression must be a non-date.

arithmetic-expression

Can be any arithmetic expression, as defined in "Arithmetic Expressions" on page 233.

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.

A year-last date field must not be specified in the arithmetic expression.

ROUNDED Phrase

For a discussion of the ROUNDED phrase, see “ROUNDED Phrase” on page 266.

SIZE ERROR Phrases

For a discussion of the SIZE ERROR phrases, see “SIZE ERROR Phrases” on page 266.

END-COMPUTE Phrase

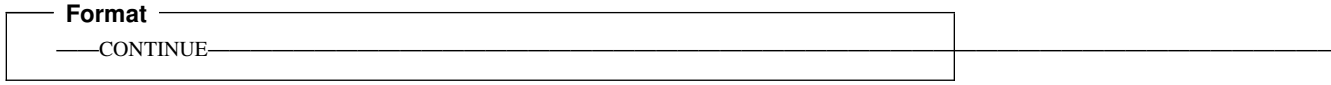
This explicit scope terminator serves to delimit the scope of the COMPUTE statement. END-COMPUTE permits a conditional COMPUTE statement to be nested in another conditional statement. END-COMPUTE can also be used with an imperative COMPUTE statement.

For more information, see “Delimited Scope Statements” on page 263.

CONTINUE Statement

The CONTINUE statement allows you to specify a no operation statement. CONTINUE indicates that no executable instruction is present.

Format

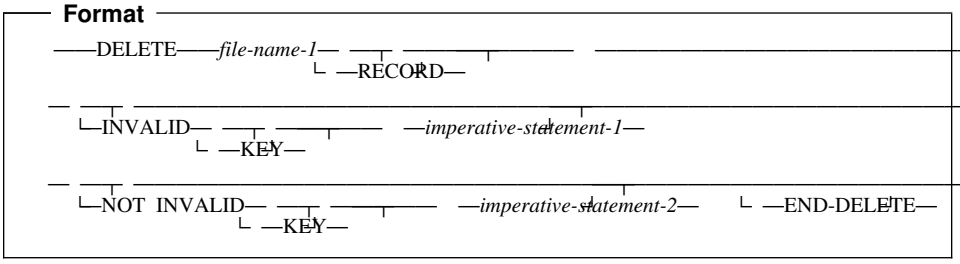


DELETE Statement

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.

OS/2 For OS/2 VSAM files, 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. OS/2

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.

DELETE Statement

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 270.)

As an IBM extension, the INVALID KEY phrase does not need to be specified for a DELETE statement that references a file in random or dynamic access and for which 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 263.

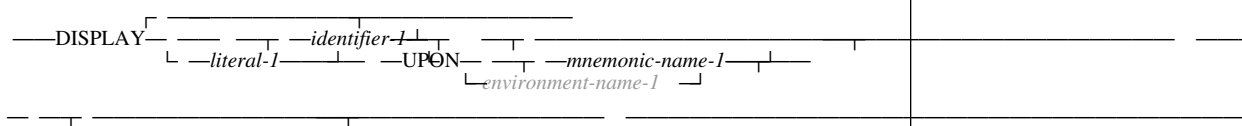
DISPLAY Statement

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.

Workstation Under AIX, OS/2, and Windows, 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. Workstation

Format



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.

Internal floating-point numbers are converted to external floating-point numbers for display, such that:

- A COMP-1 item will display as if it had an external floating-point PICTURE clause of -.9(8)E-99
- 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.

Data items defined with USAGE IS POINTER are converted to an external decimal number that would have a PICTURE clause of PIC 9(10).

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.

Date fields are treated as non-dates when specified in a DISPLAY statement. That is, the DATE FORMAT is ignored, and the content of the data item is transferred to the output device as is.

DBCS data items, explicitly or implicitly defined as USAGE DISPLAY-1, are transferred to the sending field of the output device. Under OS/390 and VM, for proper results, the output device must have the capability to recognize DBCS shift-out and shift-in control characters.

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.

Signed numeric literals and non-integer numeric literals are allowed.

Floating-point literals are allowed.

DBCS literals are allowed.

The ALL figurative constant can be used with DBCS literals in a DISPLAY verb.

UPON

mnemonic-name or **environment-name** must be associated in the SPECIAL-NAMES paragraph with an output device.

Host A default logical record size is assumed for each device, as follows:

The system logical output device = 120 characters
The system punch device = 80 characters
The console = 100 characters

A maximum logical record size is allowed for each device, as follows:

The system logical output device = 255 characters
The system punch device = 255 characters
The console = 100 characters

Note: On the system punch device, the last eight characters are used for PROGRAM-ID name. Host

When the UPON phrase is omitted, the system's logical output device is assumed. The list of valid environment-names in a DISPLAY statement is contained in Table 8 on page 91.

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.

Host Under OS/390 and VM, COBOL does not support output devices that are capable of positioning to a specific character position. See the *IBM COBOL Programming Guide* for your platform for more information about the DISPLAY statement. Host

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.

If a DBCS operand must be split across multiple records, it will be split only on a double-byte boundary.

Host Under OS/390 and VM, the shift code compensation is required under this case. That is, when a DBCS operand is split across multiple records, the shift-in character needs to be inserted at the end of the current record, and the shift-out character needs to be inserted at the beginning of the next record. A space is padded after the shift-in character, if necessary. These additional inserted shift codes and spaces are included in the count while the compiler is calculating the number of records required.

After the last operand has been transferred to the output device, the device is reset to the leftmost position of the next line of the device.

If a DBCS data item or literal is specified in a DISPLAY verb, the size of the sending field is the total character count of all operands listed, with each DBCS character counted twice, plus the necessary shift codes for DBCS. Host

Notes:

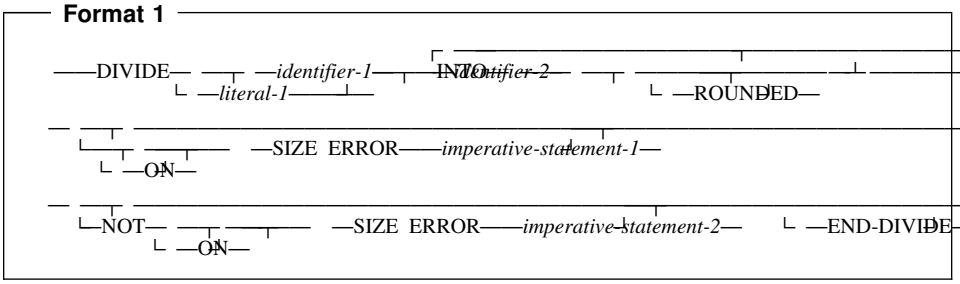
1. The DISPLAY statement causes the printer to space **before** printing.
2. The DISPLAY statement can be used to identify data records that have caused one of the following conditions:
   a. A size error
   b. An invalid key
   c. An overflow condition
   d. A status key returned as a value other than 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

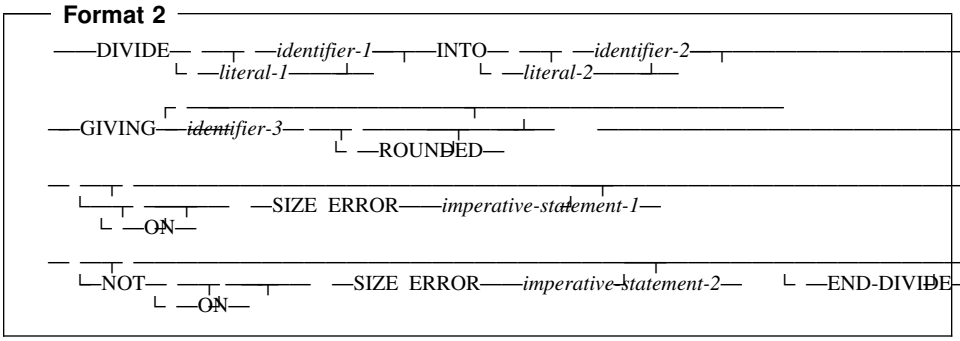
The DIVIDE statement divides one numeric data item into or by other(s) and sets the values of data items equal to the quotient and remainder.

Format 1



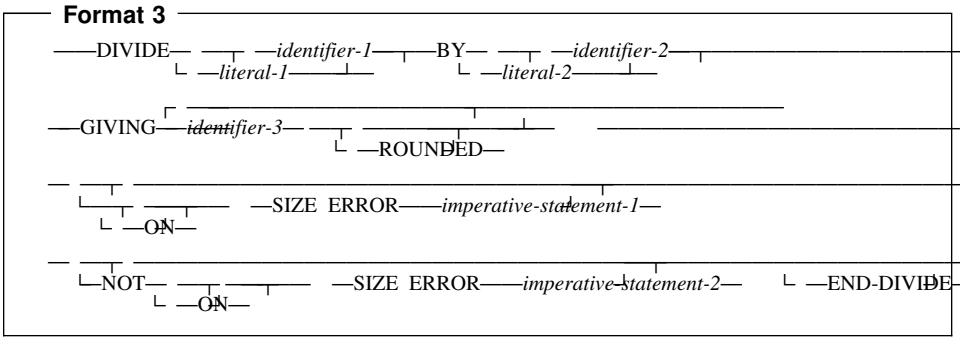
In Format 1, the value of identifier-1 or literal-1 is divided into the value of identifier-2, and the quotient is then stored in identifier-2. For each successive occurrence of identifier-2, the division takes place in the left-to-right order in which identifier-2 is specified.

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.

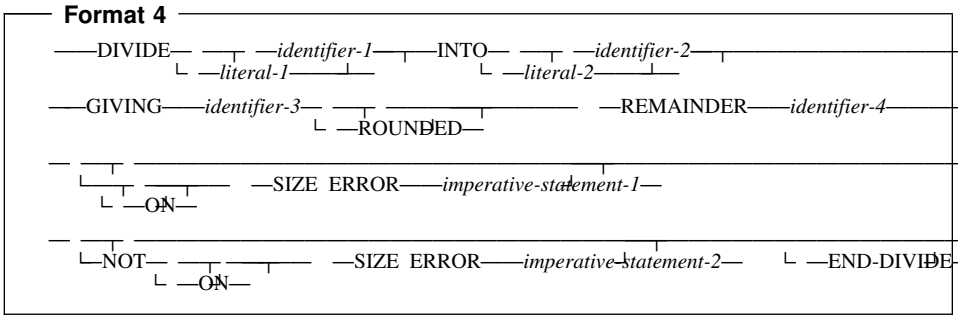
Format 3



DIVIDE Statement

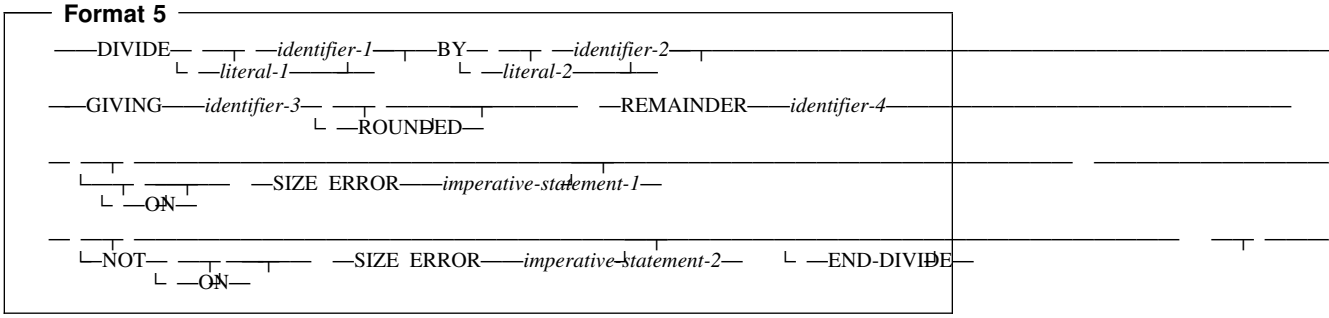
In Format 3, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2. The value of the quotient is stored in each data item referenced by identifier-3.

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-1 and identifier-2 cannot be date fields.

identifier-3, identifier-4
Must name an elementary numeric or numeric-edited item.

If identifier-3 or identifier-4 is a date field, then see "Storing Arithmetic Results That Involve Date Fields" on page 237 for details on how the quotient or remainder is stored in identifier-3.

literal-1, literal-2
Must be a numeric literal.

In Formats 1, 2, and 3, floating-point data items and literals can be used anywhere that a numeric data item or literal can be specified.

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 266.

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.

The REMAINDER phrase is invalid if the receiver or any of the operands is a floating-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 266.

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 268.

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 263.

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 227.
- Nested program. See "Nested Programs" on page 66 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 77).

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

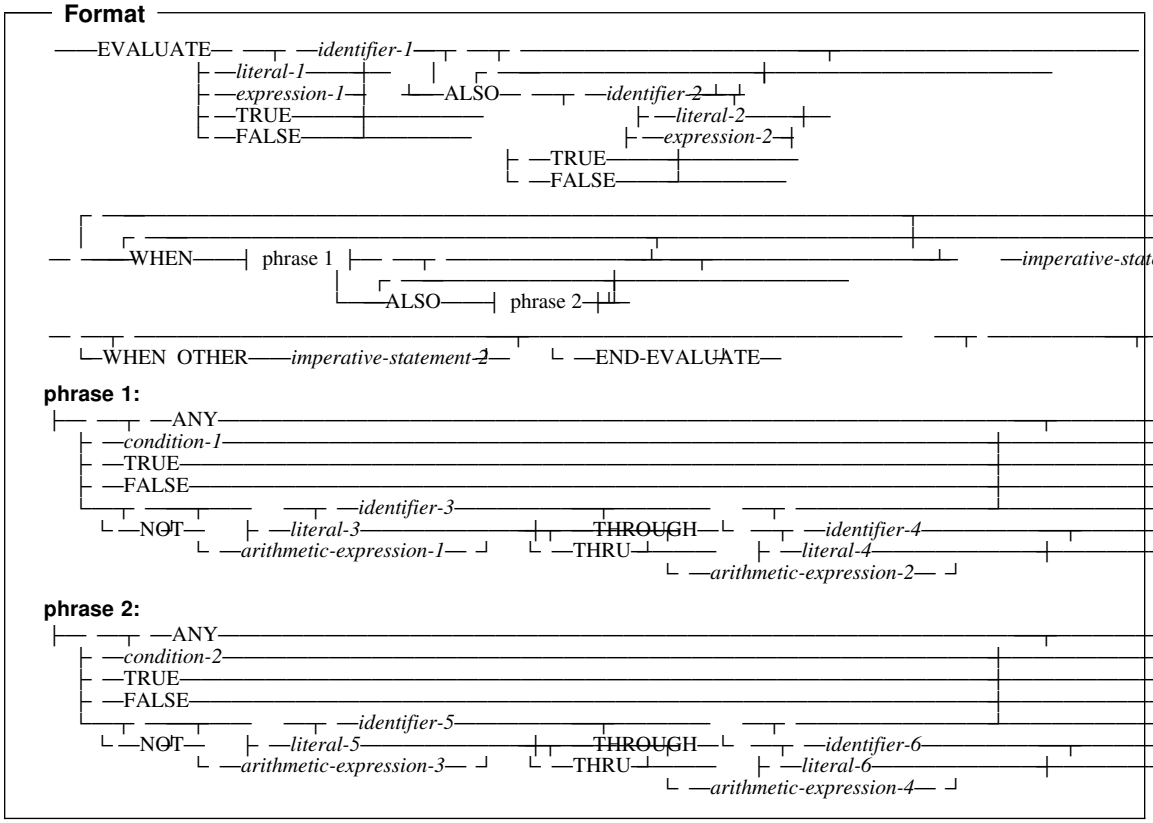
Do not specify the ENTRY statement in a program that contains a Procedure Division ...RETURNING phrase.

For a discussion of the USING phrase, see "The Procedure Division Header" on page 227.

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. For comparisons involving date fields, see "Date Fields" on page 244.

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.

Where identifiers are permitted, they can reference date field, DBCS, floating-point, USAGE POINTER, USAGE PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE or USAGE PROCEDURE-POINTER data items.

Where nonnumeric literals are permitted, DBCS literals are permitted.

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 263.

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 233.)

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 239.)

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.

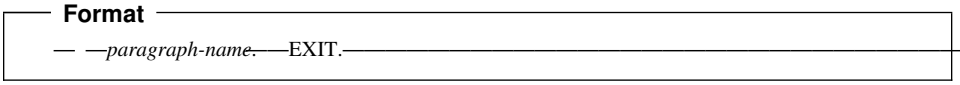
If no WHEN phrase is selected and no WHEN OTHER phrase is specified, execution continues with the next executable statement following the scope delimiter.

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.

As an IBM extension, the EXIT statement does not need to appear in a sentence by itself. Any statements following the EXIT statement are executed; the EXIT statement 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

```
──EXIT PROGRAM.──────────────────────────────────────────────────────────
```

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.

As an IBM extension, the EXIT PROGRAM statement does not have to be the last statement in a sequence of imperative statements, but the statements following the 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. It must not be used in a declarative procedure in which the GLOBAL phrase is specified.

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 application to end.)	Return to calling program.

GO TO Statement

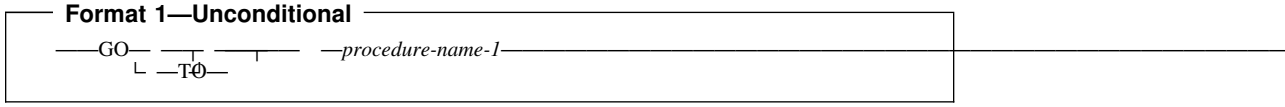
The GO TO statement transfers control from one part of the Procedure Division to another. The types of GO TO statements are:

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 285.)

Format 1—Unconditional



procedure-name-1
Must name a procedure or a section in the same Procedure Division as the GO TO statement.

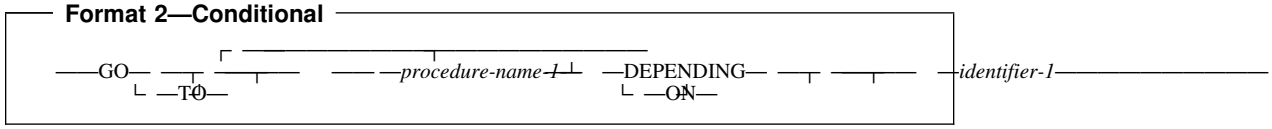
As an IBM extension, the unconditional GO TO statement does not have to be the last statement in a sequence of imperative statements. However, any statements following the GO TO are not executed.

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. Identifier-1 cannot be a windowed date field.

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

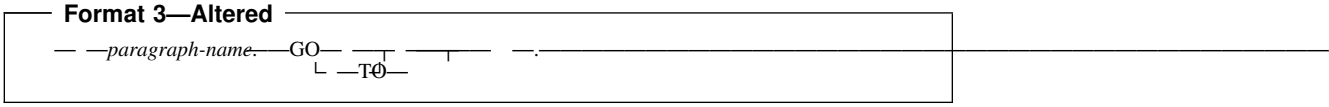
The altered GO TO statement transfers control to the first statement of the paragraph named in the ALTER statement.

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

Workstation Under AIX, OS/2, and Windows, GO TO MORE-LABELS is treated as a comment. Workstation

The GO TO MORE-LABELS statement can only be specified in a LABEL declarative.

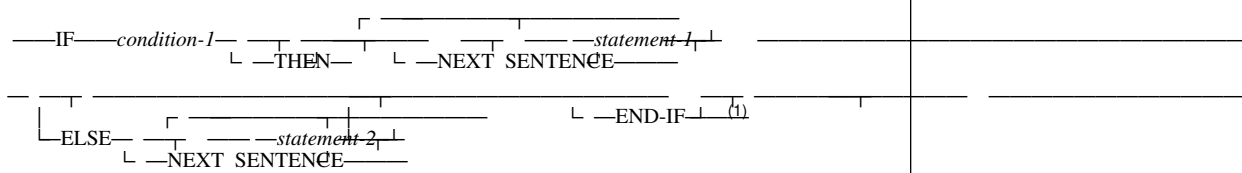
Format 4—MORE-LABELS

For more details, see the *IBM COBOL for OS/390 & 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:
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 239.

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.

END-IF can be specified with NEXT SENTENCE. However, if the NEXT SENTENCE phrase is executed, control will not pass to the next statement following the 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 263.

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

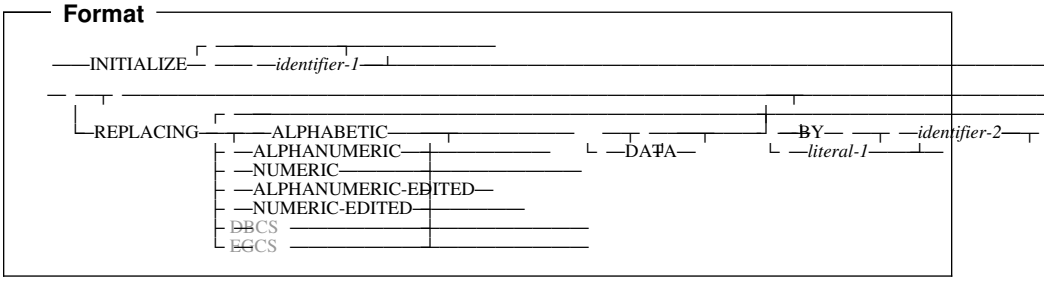
The INITIALIZE statement sets selected categories of data fields to predetermined values. It is functionally equivalent to one or more MOVE statements.

When the REPLACING phrase is not used:

SPACE is the implied sending field for alphabetic, alphanumeric, alphanumeric-edited, and DBCS items.

ZERO is the implied sending field for numeric and numeric-edited items.

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 data description entry for identifier-1 or any items subordinate to identifier-1 cannot contain the DEPENDING ON phrase of the OCCURS clause. The data description entry for 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. A floating-point data item or floating-point literal will be treated as if it is in the 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 140.

DBCS
EGCS
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

   Object references

   Data items defined with USAGE IS POINTER or USAGE IS 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

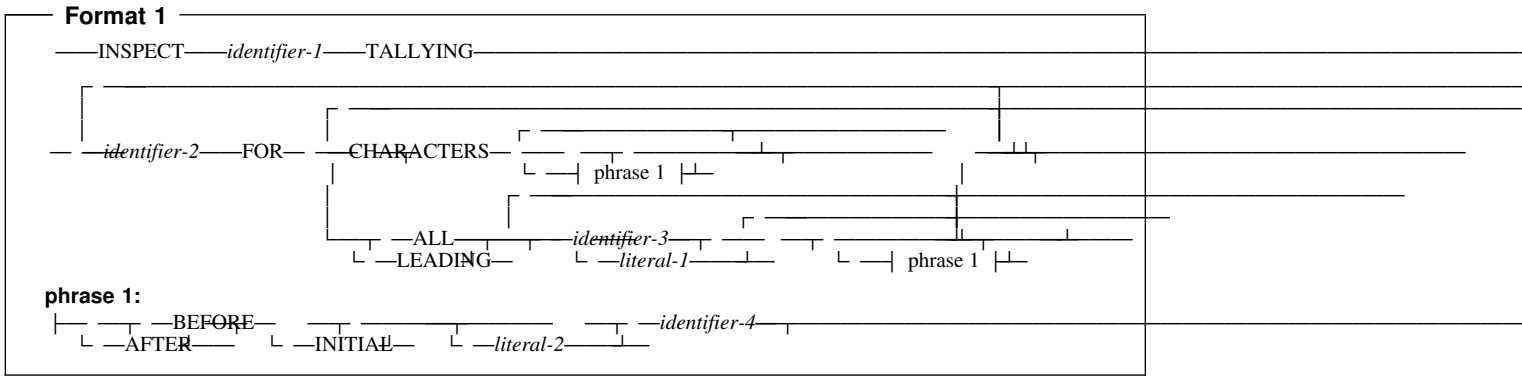
The INSPECT statement specifies that characters, or groups of characters, in a data item are to be counted (tallied) or replaced or both.

It counts the occurrence of a specific character (alphabetic, numeric, or special character) in a data item. (Formats 1 and 3)

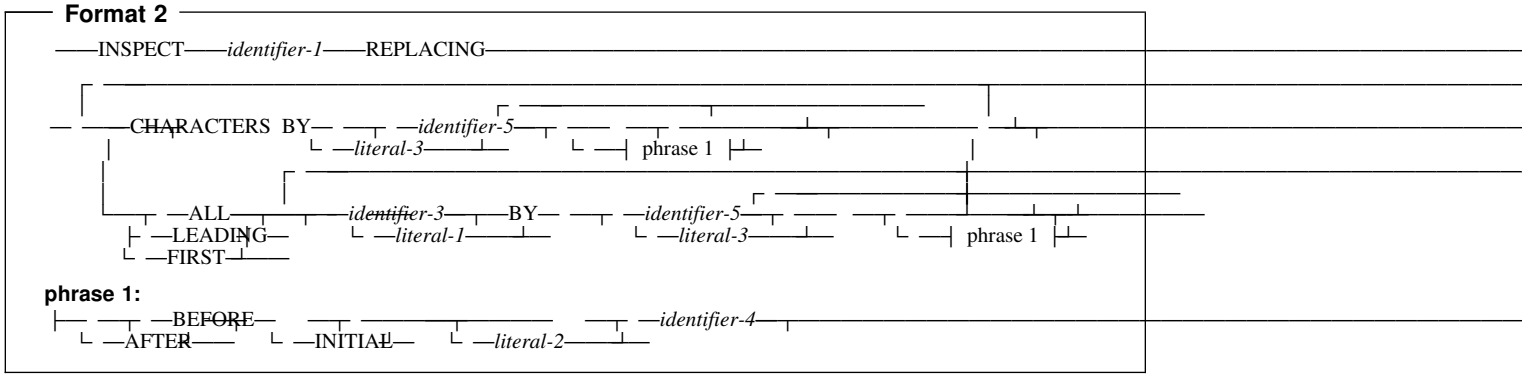
It fills all or portions of a data item with specified characters, such as spaces or zeros. (Formats 2 and 3)

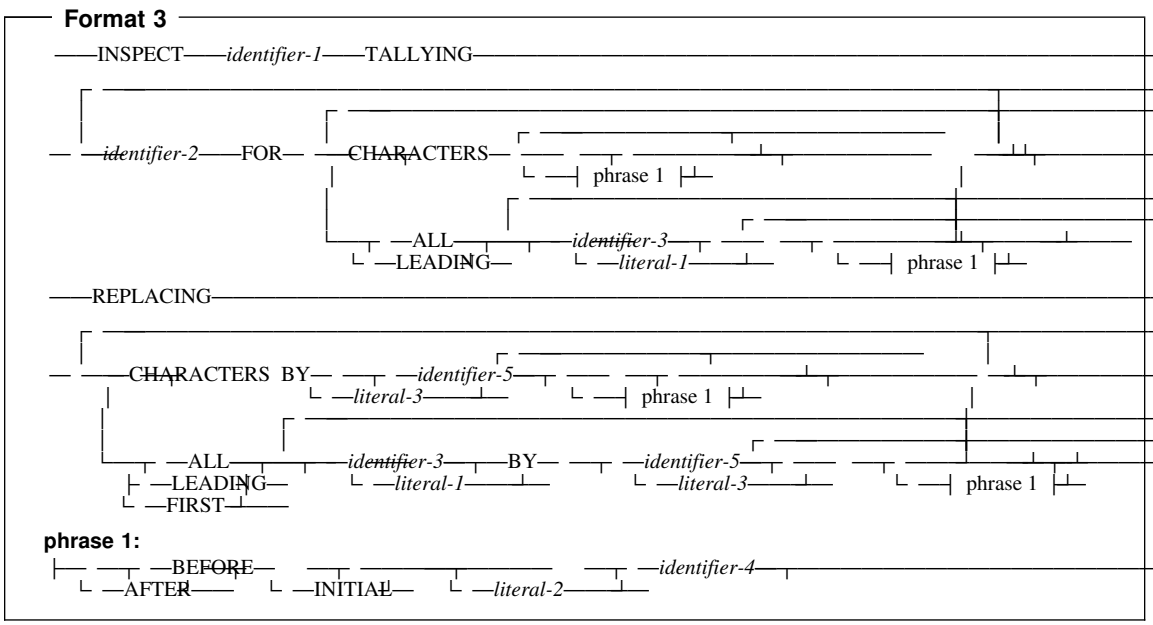
It converts all occurrences of specific characters in a data item to user-supplied replacement characters. (Format 4)

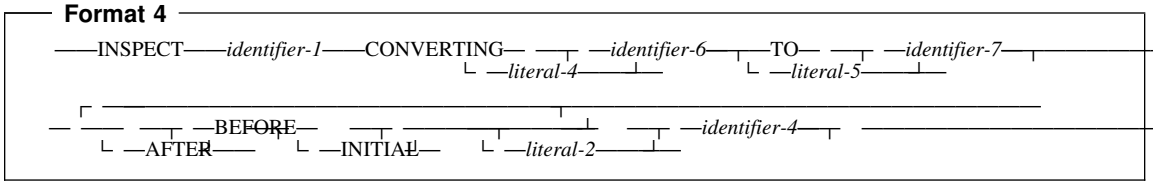
Format 1



Format 2







None of the identifiers in an INSPECT statement can be windowed date fields.

identifier-1
Is the **inspected item** and can be any 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 left-most 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
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
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
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
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 41. 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. 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 332.

Example of the INSPECT Statement

Figure 10 on page 336 is an example of INSPECT statement results.

INSPECT Statement

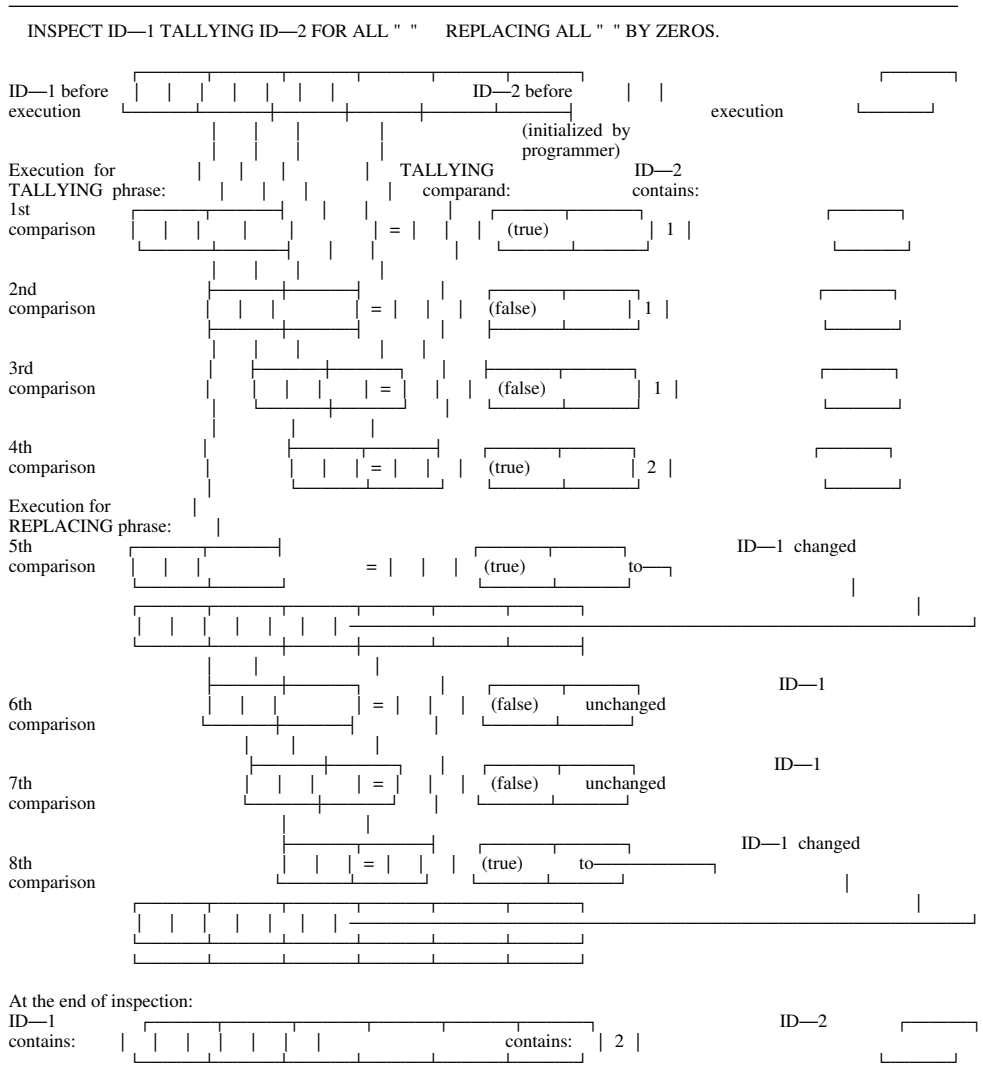


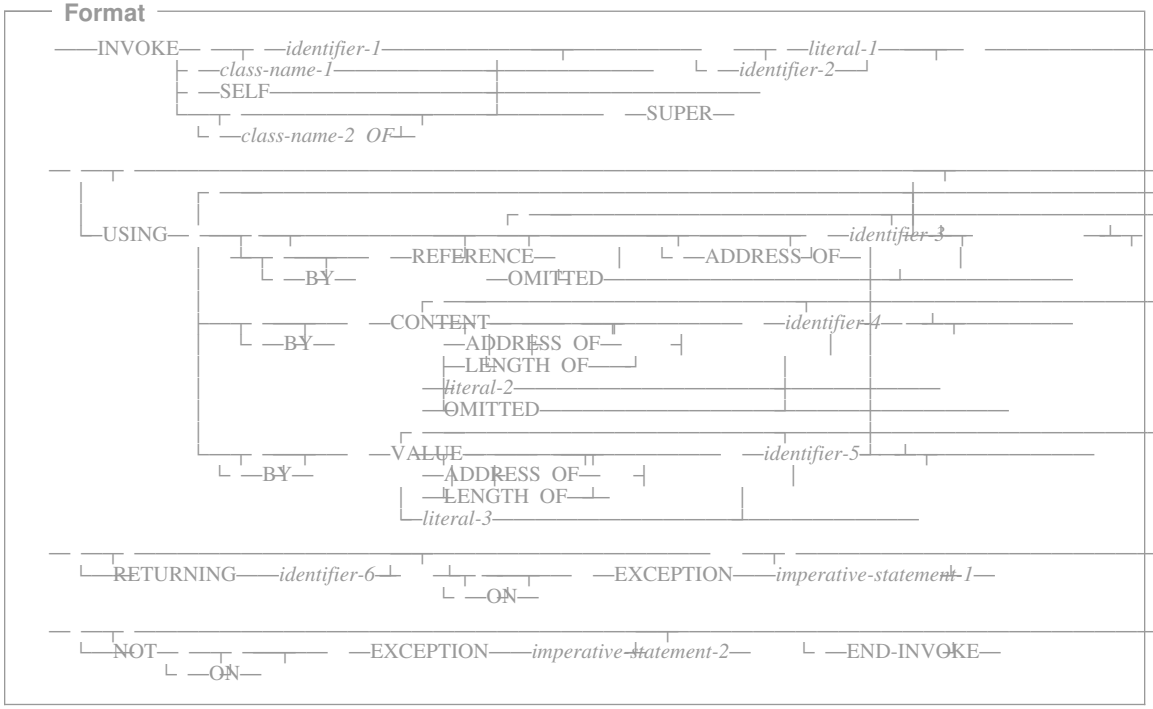
Figure 10. Example of INSPECT Statement Execution Results

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.

Identifier-2 cannot be a windowed date field.

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 227.

For conformance requirements for the USING phrase, see “Conformance Requirements for USING Phrase” on page 341.

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 10.

OMITTED
Indicates that no argument is passed.

BY CONTENT Phrase

If the BY CONTENT phrase is 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 10.

LENGTH OF Special Register
See “LENGTH OF” on page 12.

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 10.

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 12.

literal-3
Must be one of the following:

Numeric literal
ZERO
1-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.

Conformance Requirements for USING Phrase

The arguments specified on the USING phrase and the formal parameters specified on the method PROCEDURE-DIVISION USING must satisfy the following:

An invoked COBOL method must have the same number of formal parameters on its procedure division USING phrase as there are arguments on the INVOKE USING phrase. The presence or absence of USING...BY VALUE must be consistent on the INVOKE statement and the Procedure Division header of the target method.

If a formal parameter is a COBOL elementary data item not described with USAGE IS OBJECT REFERENCE, then the corresponding argument 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 PICTURE clause currency symbols can differ.

If a formal parameter is a COBOL elementary data item described with USAGE IS OBJECT REFERENCE then:

- If the argument is passed BY REFERENCE, then the argument and the parameter must be defined with identical USAGE clauses.
- If the argument is passed BY VALUE or BY CONTENT, and the parameter specified on the method Procedure Division USING phrase is a universal object reference, then the argument can be any object reference.
- If the argument is passed BY VALUE or BY CONTENT, and the parameter specified on the method Procedure Division USING phrase is an object reference typed to a specific class, then the argument must be an object reference 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: The INVOKE... RETURNING data item is an output-only parameter. On entry to the called method, the initial state of the PROCEDURE DIVISION RETURNING data item has an undefined and unpredictable value. You must initialize the PROCEDURE DIVISION RETURNING data item in the invoked method before you reference its value. The value that is passed back to the invoker is the final value of the PROCEDURE DIVISION RETURNING data item when the invoked 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 PICTURE clause currency symbols 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 341

For the RETURNING phrase, see “Conformance Requirements for RETURNING Phrase” on page 342

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 344.

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.
 1 anA USAGE OBJECT REFERENCE A.
 1 aB    USAGE OBJECT REFERENCE B.
 1 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.
 1 aP    USAGE OBJECT REFERENCE P.
 1 aQ    USAGE OBJECT REFERENCE Q.
 1 anR USAGE OBJECT REFERENCE R.
   ⋮
PROCEDURE DIVISION USING BY REFERENCE aP
                            BY  VALUE        aQ
                        RETURNING            anR.
```

In the above examples:

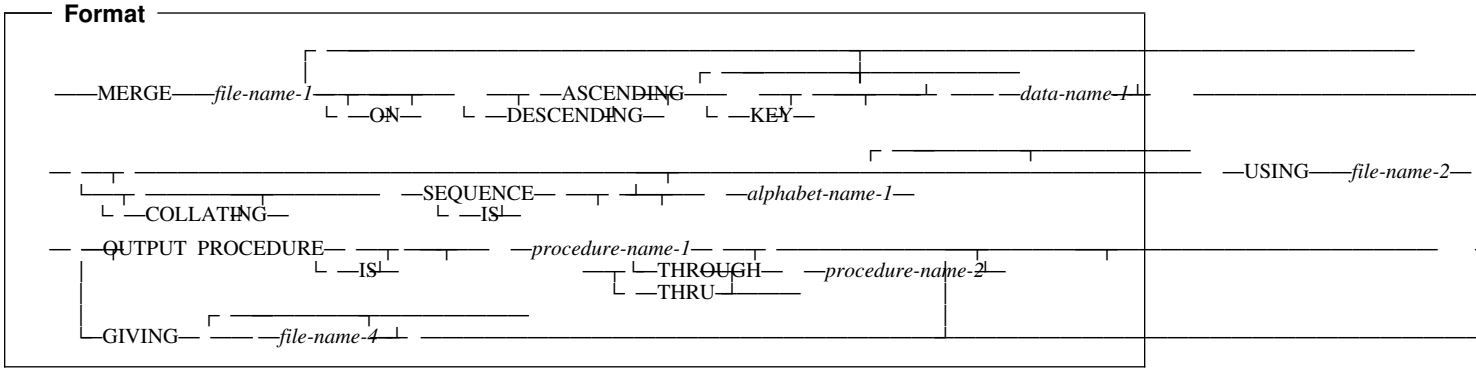
Class P and class A must be the same class.
Class Q must be the same class or a parent of B.
Class R must be the same class or a subclass of C.

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.

As an IBM extension, any file-names in a MERGE statement can be specified in 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.

KEY data items can be floating-point items.

KEY data items cannot be variably-located.

Workstation Under AIX, OS/2, and Windows, KEY data items cannot be windowed date fields. Workstation

Host Under OS/390 and VM, KEY data items can be windowed date fields, under these conditions:

- The input files specified in the USING phrase may be sequential, relative, or indexed, but must not have any RECORD KEY, ALTERNATE RECORD KEY, or RELATIVE KEY in the same position as a windowed date merge key. The file system does not support windowed date fields as keys, so any ordering imposed by the file system could conflict with the windowed date field support for the merge operation. In fact, if the merge is to succeed, then input files must have already been sorted into the same order as that specified by the MERGE statement, including any windowed date ordering.
- The GIVING phrase must not specify an indexed file, because the (binary) ordering assumed or imposed by the file system conflicts with the windowed date ordering provided in the output of the merge. Attempting to write the windowed date merge output to such an indexed file will either fail or re-impose binary ordering, depending on how the file is accessed (the ACCESS MODE in the file-control entry).
- If an alphanumeric windowed date field is specified as a KEY for a MERGE statement, the collating sequence in effect for the merge operation must be EBCDIC. Thus the COLLATING SEQUENCE phrase of the MERGE statement or, if this phrase is not specified, then any PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph, must not specify a collating sequence other than EBCDIC or NATIVE.

If the MERGE statement meets these conditions, then the merge operation takes advantage of SORT Year 2000 features, assuming that the execution environment includes a sort product that supports century windowing.

A year-last windowed date field can be specified as a KEY for a MERGE statement, and can thereby exploit the corresponding century windowing capability of the sort product.

For more information on using windowed date fields as KEY data items, see the *IBM COBOL Programming Guide* for your platform. Host

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" below).

Host Under OS/390 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. Host

Workstation Under AIX, OS/2, and Windows, if the KEY is a DBCS item, then the sequence of the KEY values is based on a collation sequence according to the COLLSEQ compiler option:

- If the COLLSEQ(NATIVE) compiler option is in effect, then the collating sequence is determined by the locale. For information on the locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 568.
- Otherwise, the collating sequence is determined by the binary values of the DBCS characters. Workstation

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 243).

COLLATING SEQUENCE Phrase

This phrase specifies the collating sequence to be used in nonnumeric comparisons for the KEY data items in this merge operation.

Workstation Under AIX, OS/2, and Windows, the COLLATING SEQUENCE phrase is only valid when an ASCII code page is in effect. Workstation

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
Host Under OS/390 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 548.) Host

Workstation Under AIX, OS/2, and Windows, the collating sequence is based on the character's hex value order. Workstation

STANDARD-2
Host Under OS/390 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. Host

Workstation Under AIX, OS/2, and Windows the collating sequence is based on the character's hex value order. Workstation

NATIVE
Host Under OS/390 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 548.) Host

Workstation Under AIX, OS/2, and Windows, the collating sequence indicated by the locale is selected. Workstation

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 548.)

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,

Host Under OS/390 and VM, the EBCDIC collating sequence is used. (See Appendix B, "EBCDIC and ASCII Collating Sequences" on page 548.) Host

Workstation Under AIX, OS/2, 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 568.

Workstation

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.

procedure-name-1
Specifies the first (or only) section or paragraph in the OUTPUT PROCEDURE.

procedure-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 317).

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 16.

SORT-MESSAGE Special Register (OS/390 and VM Only)

For information, see “SORT-MESSAGE” on page 18.

Host The special register SORT-MESSAGE is equivalent to an option control statement key word in the sort control file. Host

SORT-RETURN Special Register
For information, see “SORT-RETURN” on page 18.

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.

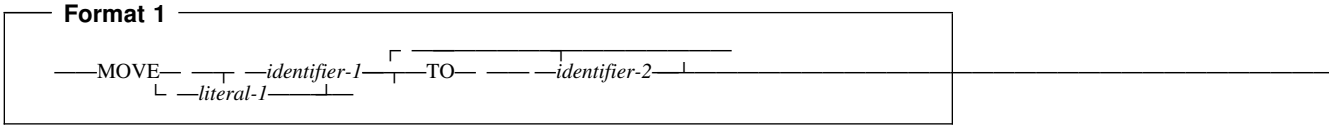
If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

1. Totally within non-independent segments, or
2. Wholly within the same independent segment as that MERGE statement.

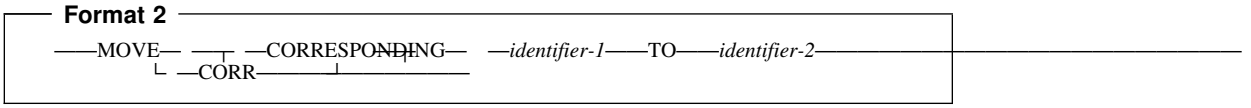
MOVE Statement

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 353 and “Group Moves” on page 356.

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 on page 265. The results are the same as if each pair of CORRESPONDING identifiers were referenced in a separate MOVE statement.

Do not specify a data item defined with USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE in a MOVE statement.

A data item defined with USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE can be part of a group that is referred to in a MOVE 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 172).

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.

Floating-point—includes internal floating-point items (defined as USAGE COMP-1 or USAGE COMP-2), external floating-point items (defined as USAGE DISPLAY), and floating-point literals.

DBCS—includes DBCS data items (defined explicitly or implicitly as USAGE DISPLAY-1) and DBCS literals.

MOVE Statement

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 141.

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 141.

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 141.

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.

When the sending item is floating-point, the data is first converted to either a binary 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.

Floating-point:

The sending item is converted first to internal floating-point and then moved.

When data is moved to or from an external floating-point item, the data is converted 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.

Workstation 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). Workstation

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 42 shows valid and invalid elementary moves for each category. In the table:

YES = Move is valid.
NO = Move is invalid.

Table 42. Valid and Invalid Elementary Moves

	Receiving Item Category							
Sending Item Category	**Alpha-betic**	**Alpha-numeric**	**Alpha-numeric 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.

Moves Involving Date Fields

If the sending item is specified as a year-last date field, then all receiving fields must also be year-last date fields with the same date format as the sending item. If a year-last date field is specified as a receiving item, then the sending item must be either a non-date or a year-last date field with the same date format as the receiving item. In both cases, the move is then performed as if all items were non-dates.

Table 43 describes the behavior of moves involving non-year-last date fields. If the sending item is a date field, then the receiving item must be a compatible date field. If the sending and receiving items are both date fields, then they must be compatible; that is, they must have the same date format, except for the year part, which may be windowed or expanded.

This table uses the following terms to describe the moves:

Normal
The move is performed with no date-sensitive behavior, as if the sending and receiving items were both non-dates.

Expanded
The windowed date field sending item is treated as if it were first converted to expanded form, as described under "Semantics of Windowed Date Fields" on page 165.

Invalid
The move is not allowed.

Table 43. Moves Involving Date Fields

	Receiving Item		
Sending Item	**Non-date**	**Windowed date field**	**Expanded date field**
Non-date	Normal	Normal	Normal
Windowed date field	Invalid	Normal	Expanded
Expanded date field	Invalid	Normal[1]	Normal

1 A move from an expanded date field to a windowed date field is, in effect, a "windowed" move, because it truncates the century component of the expanded date field. If the move is alphanumeric, it treats the receiving windowed date field as if its data description specified JUSTIFIED RIGHT. This is true even if the receiving windowed date field is a group item, for which the JUSTIFIED clause cannot be specified.

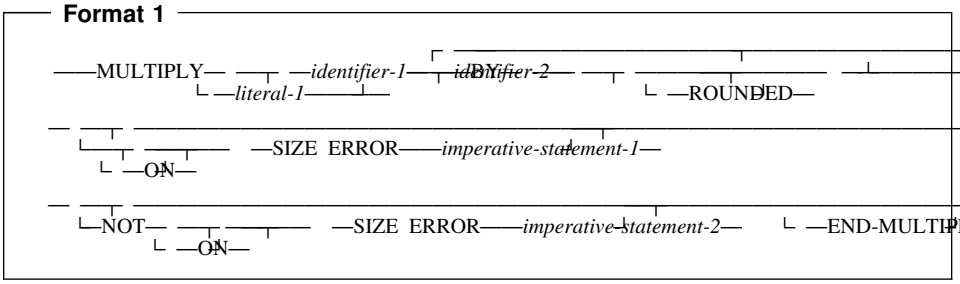
Group Moves

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 172.) **All** group moves are valid.

MULTIPLY Statement

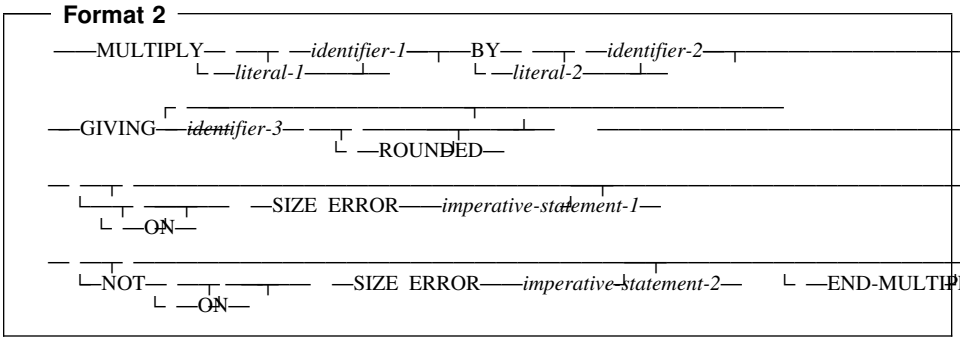
The MULTIPLY statement multiplies numeric items and sets the values of data items equal to the results.

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. Identifier-1 and identifier-2 cannot be date fields.

literal-1, literal-2
Must be a numeric literal.

MULTIPLY Statement

For Format-2:

identifier-3

Must name an elementary numeric or numeric-edited item.

Identifier-3, the GIVING phrase identifier, is the only identifier in the MULTIPLY statement that can be a date field.

If identifier-3 names a date field, then see "Storing Arithmetic Results That Involve Date Fields" on page 237 for details on how the product is stored in identifier-3.

Floating-point data items and literals can be used anywhere a numeric data item or literal can be specified.

The composite of operands must not contain more than 18 digits.

The composite of operands can be more than 18 digits. For information on arithmetic intermediate results, see the *IBM COBOL Programming Guide* for your platform.

ROUNDED Phrase

For Formats 1 and 2, see "ROUNDED Phrase" on page 266.

SIZE ERROR Phrases

For Formats 1 and 2, see "SIZE ERROR Phrases" on page 266.

END-MULTIPLY Phrase

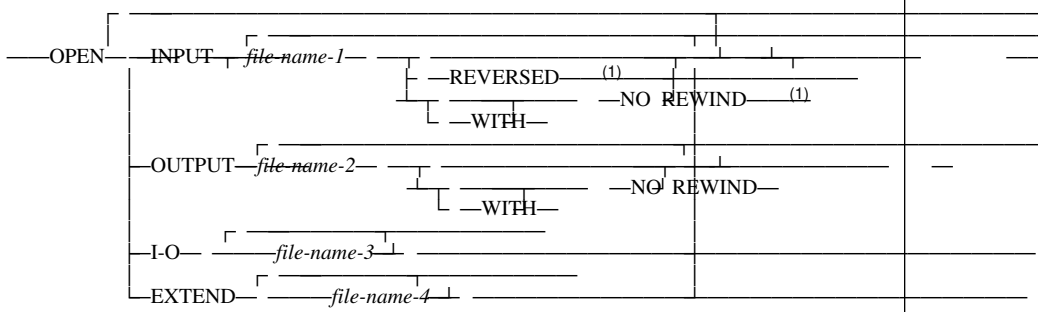
This explicit scope terminator serves to delimit the scope of the MULTIPLY statement. END-MULTIPLY permits a conditional MULTIPLY statement to be nested in another conditional statement. END-MULTIPLY can also be used with an imperative MULTIPLY statement.

For more information, see "Delimited Scope Statements" on page 263.

OPEN Statement

The OPEN statement initiates the processing of files. It also checks and/or writes labels.

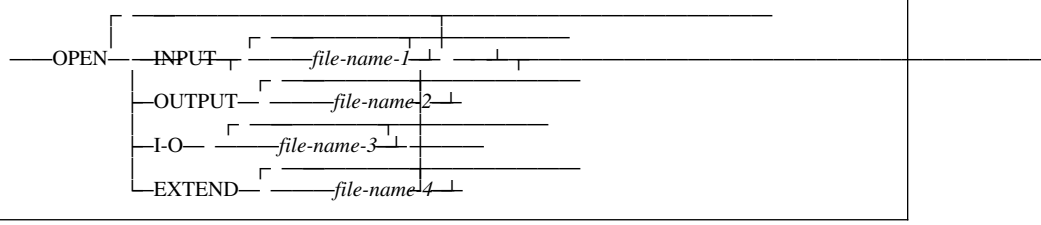
Format 1—Sequential Files



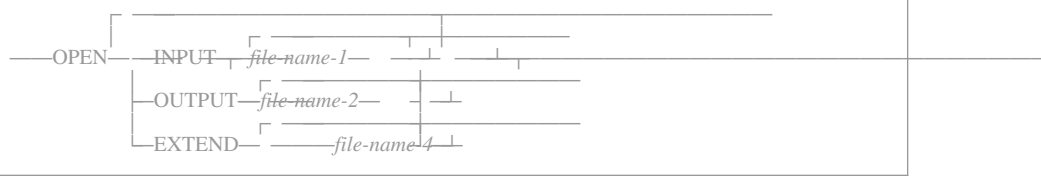
Note:

[1] Under OS/390, 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 OS/390, 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.

Workstation Under AIX, OS/2, and Windows, the I-O phrase is not valid for line sequential files. Workstation

EXTEND
Permits opening the file for output operations.

The EXTEND phrase is only allowed for sequential access files if the new data is written in ascending sequence. As an IBM extension, the EXTEND phrase is allowed for files that specify the LINAGE clause.

Under OS/390, for QSAM files, do not specify the EXTEND phrase for a multiple file reel.

If you want to append to a file, but are unsure if the file exists, use the SELECT OPTIONAL clause before OPENing the file in EXTEND mode. The file will be created or appended to, depending on whether the file exists.

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
Host Under OS/390 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. Host

Workstation Under AIX, OS/2, and Windows, the REVERSED phrase is treated as a comment. Workstation

NO REWIND

Host Under OS/390 and VM, the NO REWIND phrase is only valid for sequential single reel files. It is not valid for VSAM files. Host

Workstation Under AIX, OS/2, and Windows, the NO REWIND phrase is treated as a comment. Workstation

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.

OS/2 Any OS/2 VSAM files, that specify the LOCK MODE IS AUTOMATIC phrase are shareable (except for any VSAM files residing on a LAN server). 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.” OS/2

Workstation When the EXTEND phrase is not specified, OPEN statement positions the file at its beginning. Workstation

Host For OS/390 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 OS/390 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. Host

Host If the PASSWORD clause is specified in the FILE-CONTROL entry, the password data item must contain the valid password before the OPEN statement is executed. If the valid password is not present, the OPEN statement execution is unsuccessful. Host

Label Records

Workstation 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 Workstation

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 44 on page 363 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 44. 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 45 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 45. Permissible Statements for Sequential Files

Statement	Open Mode			
	Input	Output	I-O	Extend
READ	X		X	
WRITE		X		X
REWRITE			X	

Table 46 on page 364 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 46. 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 47 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 47. 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:

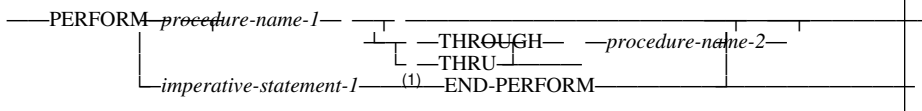
Basic PERFORM
TIMES phrase PERFORM
UNTIL phrase PERFORM
VARYING phrase PERFORM

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:
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 statement. 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.

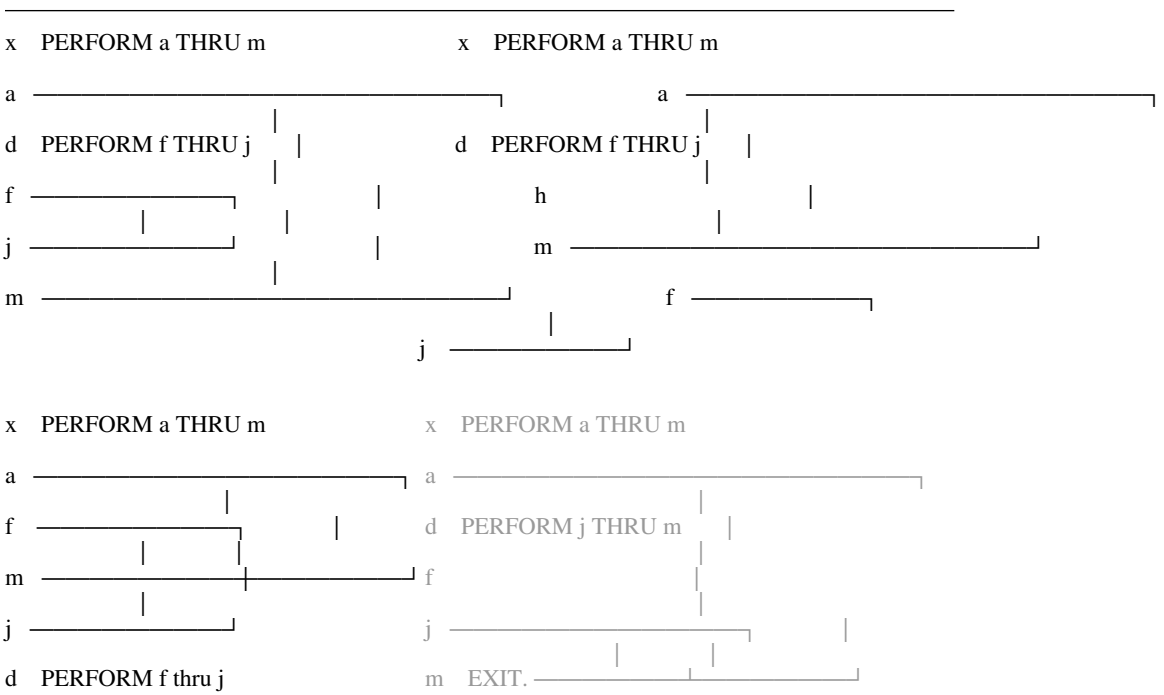


Figure 11. Valid PERFORM Statement Execution Sequences

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

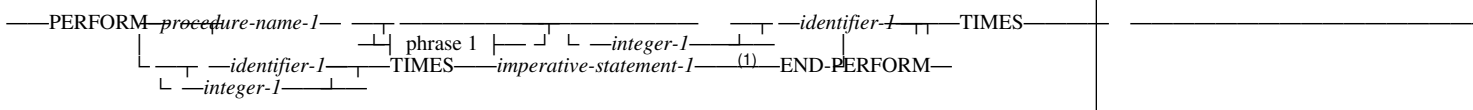
Delimits the scope of the in-line PERFORM statement. Execution of an in-line PERFORM is completed after the last statement contained within it has been executed.

PERFORM with TIMES Phrase

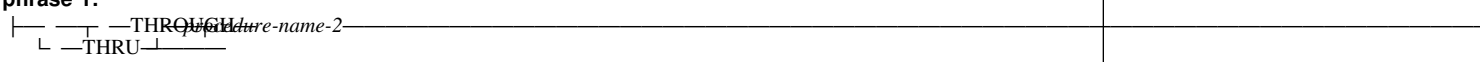
The procedure(s) referred to in the TIMES phrase PERFORM statement are executed the **number of times** specified by the value in identifier-1 or integer-1. Control then passes to the next executable statement following the PERFORM statement.

PERFORM Statement

Format 2



phrase 1:



Note:
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. Identifier-1 cannot be a windowed date field.

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.

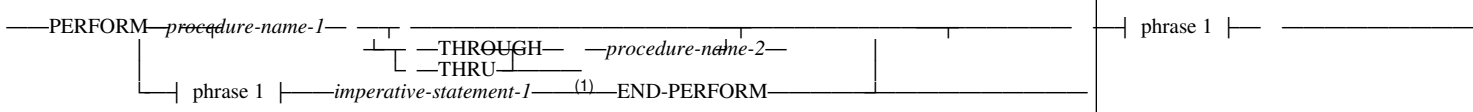
After the PERFORM statement has been initiated, any change to identifier-1 has no effect in varying the number of times the procedures are initiated.

integer-1
Can be a positive signed integer.

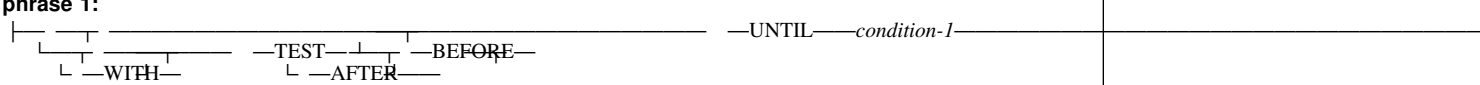
PERFORM with UNTIL Phrase

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



phrase 1:



Note:
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 239. 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).

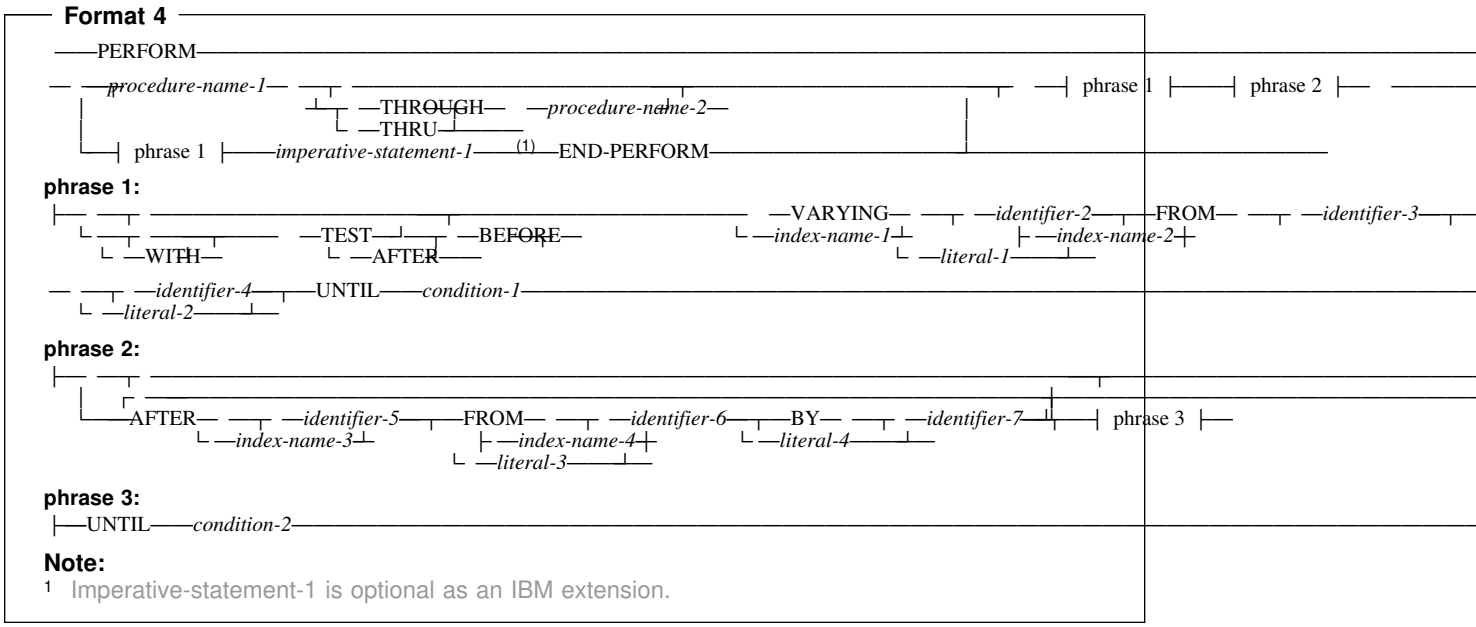
If the TEST AFTER phrase is specified, the statements to be performed are executed at least once before the condition is tested (corresponds to DO UNTIL).

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 375.)

The Format 4 VARYING phrase PERFORM statement can serially search an entire 7-dimensional table.



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. These identifiers cannot be windowed date fields.

literal-1 thru 4

Must represent a numeric literal.

condition-1, condition-2

Can be any condition described under "Conditional Expressions" on page 239. 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.

Floating-point data items and literals can be used anywhere a numeric data item or 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 data item referenced by the identifier is used in a setting or an augmenting operation.

Figure 12 on page 371 illustrates the logic of the PERFORM statement when an identifier is varied with TEST BEFORE.

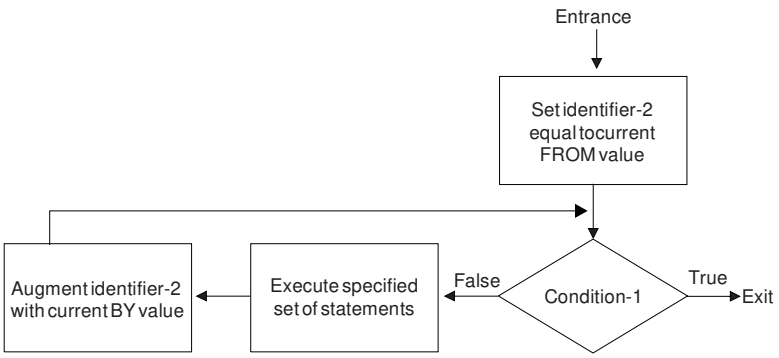


Figure 12. Varying One Identifier—with TEST BEFORE

Figure 13 illustrates the logic of the PERFORM statement when an identifier is varied with TEST AFTER.

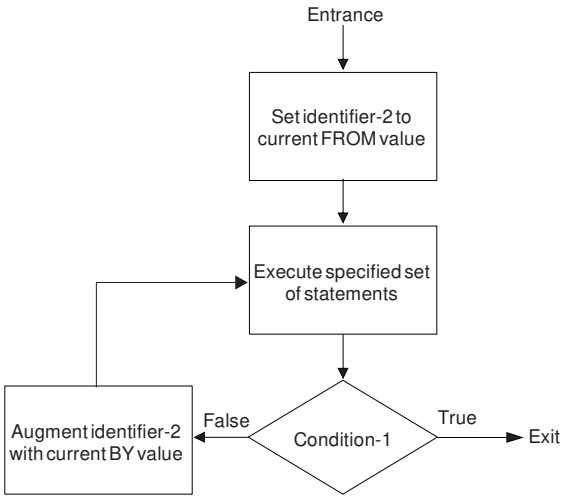


Figure 13. Varying One Identifier—with TEST AFTER

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 on page 373 illustrates the logic of the PERFORM statement when two identifiers are varied with TEST BEFORE.

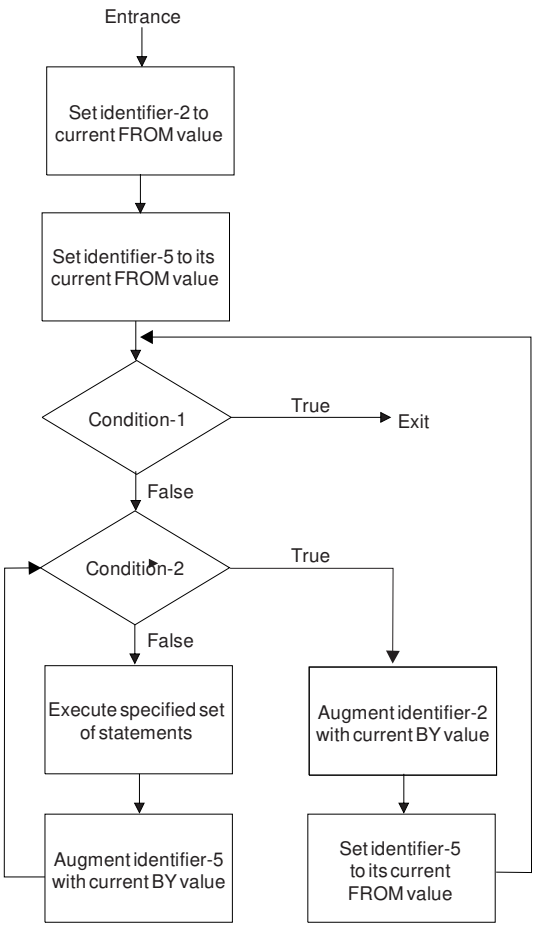


Figure 14. Varying Two Identifiers—with TEST BEFORE

Figure 15 on page 374 illustrates the logic of the PERFORM statement when two identifiers are varied with TEST AFTER.

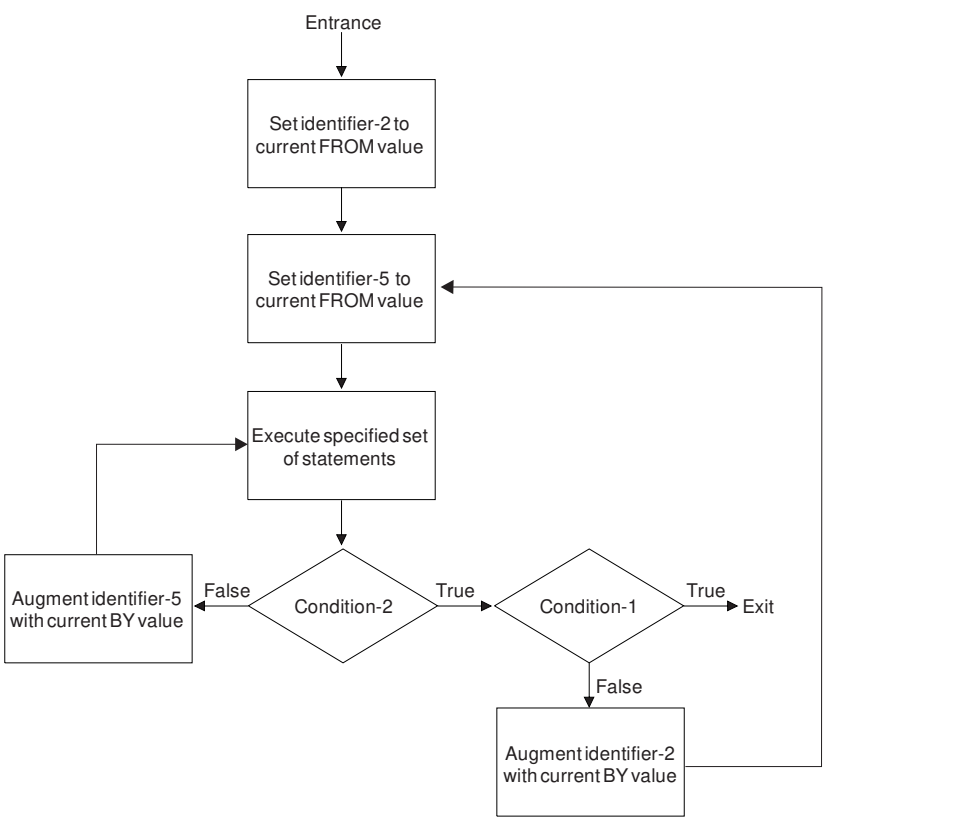


Figure 15. Varying Two Identifiers—with TEST AFTER

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-1  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:

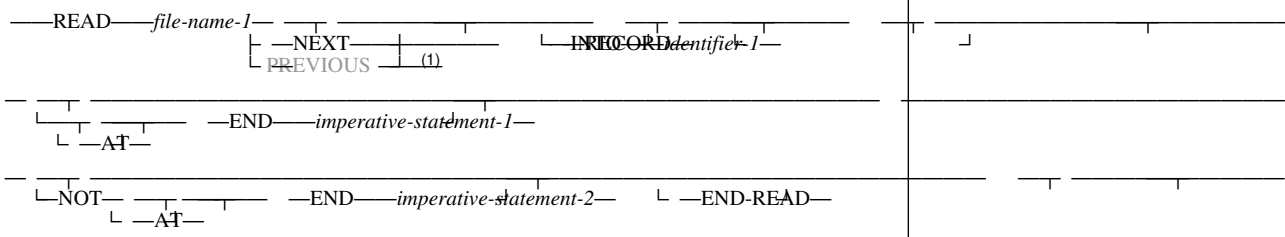
1. In the VARYING/AFTER phrases, when an index-name is specified:
   a. The index-name is initialized and incremented or decremented according to the rules under “INDEXED BY Phrase” on page 175. (See also “SET Statement” on page 400.)
   b. In the associated FROM phrase, an identifier must be described 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.

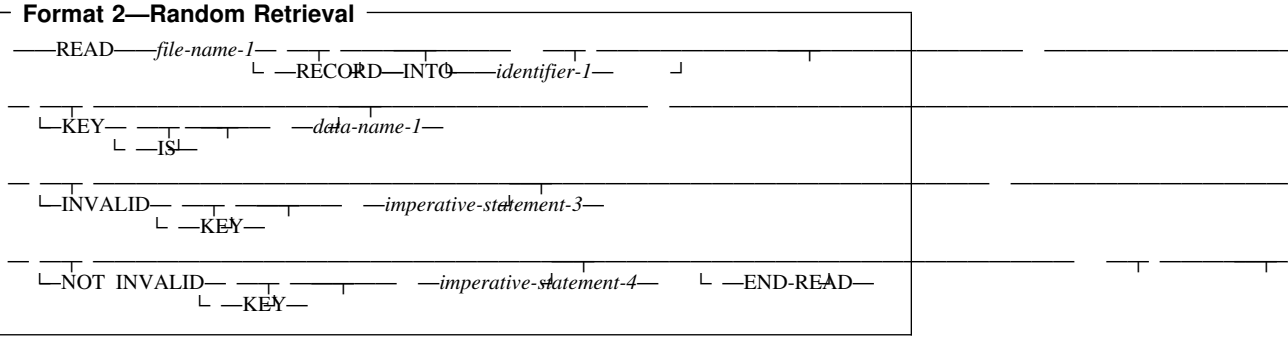
When the READ statement is executed, the associated file must be open in INPUT or I-O mode.

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
Reads 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.

Host Under OS/390 and VM, you must specify the NEXT RECORD phrase for files in dynamic access mode, which are retrieved sequentially. Host

Workstation 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. Workstation

PREVIOUS RECORD (Workstation Only)

Workstation Reads 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. Workstation

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.

If identifier-1 is a date field, then the implied MOVE statement is performed according to the behavior described under “Moves Involving Date Fields” on page 356.

The INTO phrase can be specified in a READ if:

Only one record description is subordinate to the file description entry, or

READ Statement

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.

Identifier-1 (the record area) can be a DBCS or floating point data item.

Multiple non-alphanumeric records can be specified for file-name-1. Identifier-1 need not describe a group item or an elementary alphanumeric item. The following rules apply:

1. If the file referenced by file-name-1 is described as containing variable-length records, or as a QSAM file with RECORDING MODE 'S' or 'U', a group move will take place.
2. If the file referenced by file-name-1 is described as containing fixed-length records, the movement will take place according to the rules for the MOVE statement, using, as a sending field description, the record that specifies the largest number of character positions. If more than one such record exists, the sending field record selected will be the one among those records that appears first under the description of file-name-1.

OS/2 Under OS/2, you can only READ a file with LOCK MODE AUTOMATIC 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. OS/2

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.

Data-name-1 (the record key) can be defined as a DBCS data item.

When the RECORD KEY clause specifies a DBCS data item, a KEY specified on the 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.

The AT END phrase does not have to be specified if no applicable USE AFTER STANDARD EXCEPTION procedure is specified for file-name-1.

For information on at-end condition processing, see “At End Condition” on page 381.

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.

The INVALID KEY phrase does not have to be specified if no applicable USE AFTER STANDARD EXCEPTION procedure is specified for file-name-1.

For information on INVALID KEY phrases processing, see “Invalid Key Condition” on page 274.

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 263.

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):

```
123456789 123456789
```

Contents of input area after READ is executed:

```
123456789 123456789??????????
                    |         |
                    |         |
                    └─────────┘
```

(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.

   If the AT END phrase is not specified, a USE AFTER STANDARD EXCEPTION 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 (OS/390 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.

Workstation Under AIX, OS/2, and Windows, PREVIOUS RECORD is the prior logical record in the key sequence. Workstation

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 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.

For files in dynamic access mode, the NEXT phrase (or alternatively for AIX, OS/2, and Windows, the PREVIOUS phrase) must be specified for sequential record retrieval.

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 exists), or that an optional input file is not present.

Host Under OS/390 and VM, the same procedure occurs as for sequential files (see "At End Condition" on page 381). Host

Workstation Under AIX, OS/2, and Windows see the discussion on PREVIOUS RECORD on 376. Workstation

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 381).

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.

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.

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 270.)

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 270.)

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.

READ Statement

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.

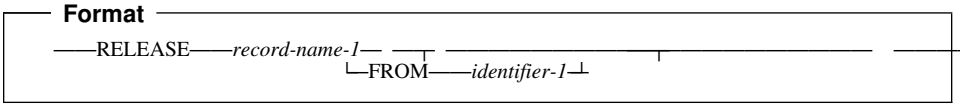
Following unsuccessful READ statement execution, the contents of the associated record area and the value of the file position indicator are undefined. Attempts to access or move data into the record area following an unsuccessful read can result in a protection exception.

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.

Can define a floating-point data item or DBCS data item. 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 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.

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 270.)

RELEASE Statement

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.

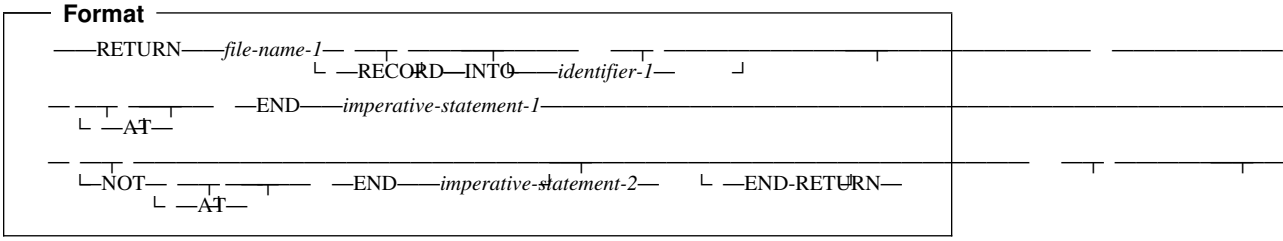
When control passes from the INPUT PROCEDURE, the sort file consists of all those records placed in it by execution of RELEASE statements.

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.

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.

Multiple non-alphanumeric records can be specified for file-name-1. Identifier-1 need not describe a group item or an elementary alphanumeric item. The following rules apply:

1. If the file referenced by file-name-1 contains variable-length records or, under OS/390 and VM, a QSAM file with RECORDING MODE 'S' or 'U', a group move will take place.
2. If the file referenced by file-name-1 contains fixed-length records, the movement will take place according to the rules for the MOVE statement, using, as a sending field description, the record that specifies the largest number of character positions. If more than one such record exists, the sending field record selected will be the 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.

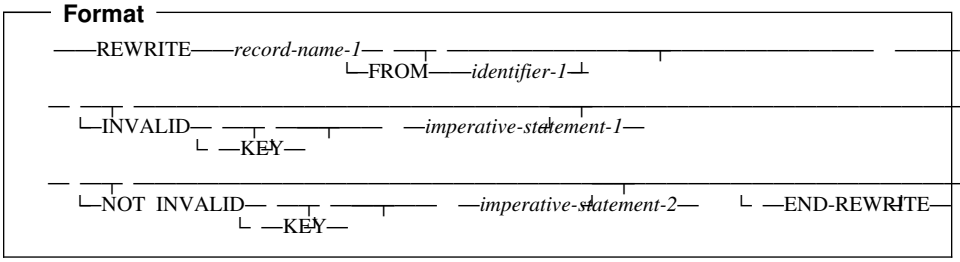
For more information, see “Delimited Scope Statements” on page 263.

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.

Workstation Under AIX, OS/2, and Windows the REWRITE statement is not supported for line sequential files. Workstation

Format



record-name-1

Must be the name of a logical record in a Data Division FD entry. The record-name can be qualified.

Record-name-1 can define a floating-point data item or DBCS data item. 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 can 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
- 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 REWRITE statement is executed, the information is still available in identifier-1 (See “INTO/FROM Identifier Phrase” under “Common Processing Facilities” on page 270).

REWRITE Statement

INVALID KEY Phrases

(See “Invalid Key Condition” under “Common Processing Facilities” on page 270.)

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 263.

Effect of Record Locking (OS/2 VSAM Files Only)

OS/2 For OS/2 VSAM files, 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. OS/2

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.

The number of character positions in record-name-1 can be different from the number 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.

The INVALID KEY phrase does not have to be specified if an applicable USE AFTER 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 270.)

Relative Files

The number of character positions in record-name-1 must equal the number of character positions in the record being replaced.

The number of character positions in record-name-1 can be different from the number 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.

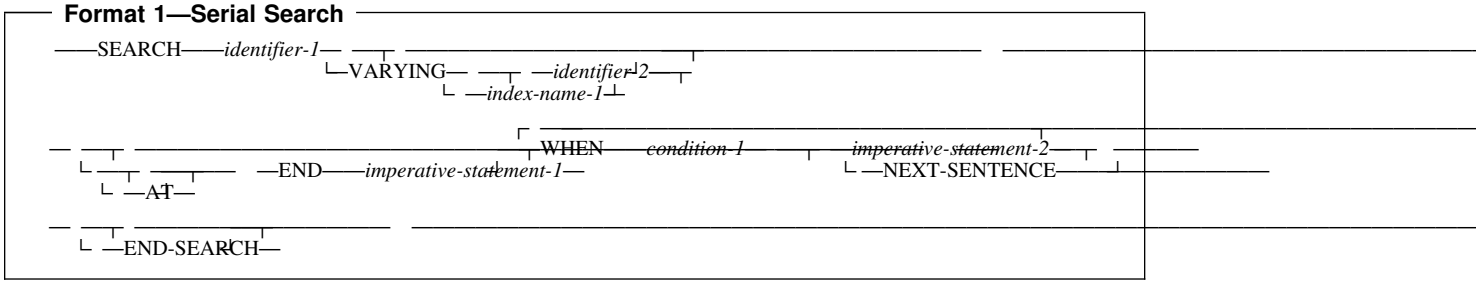
The INVALID KEY phrase does not have to be specified if an appropriate USE AFTER STANDARD EXCEPTION procedure is not specified.

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 270.) The updating operation does not take place, and the data in record-name is unaffected.

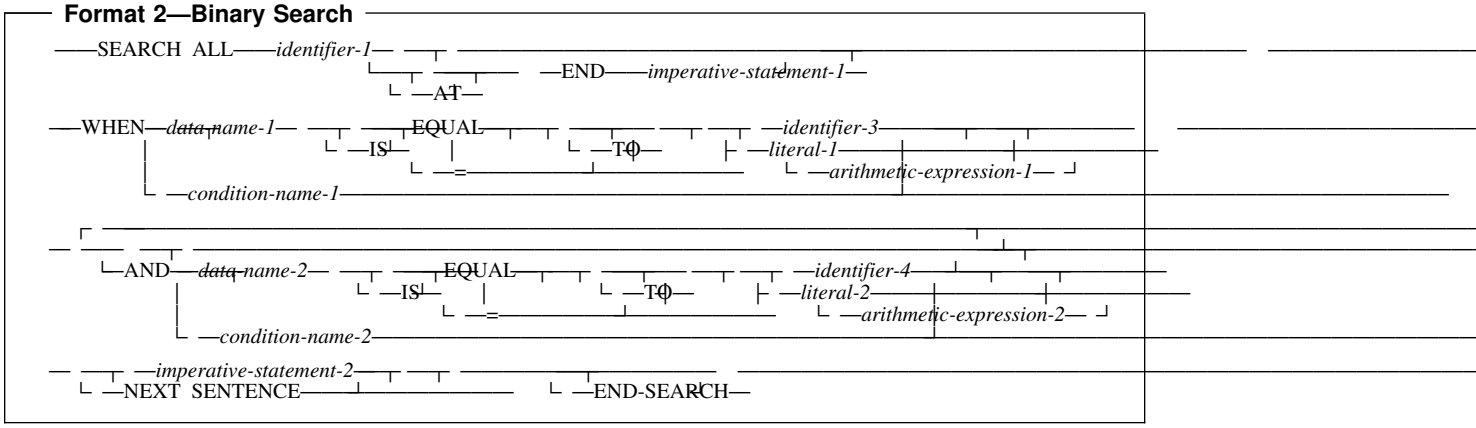
SEARCH Statement

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 2- to 7-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

As an IBM extension, you can specify END-SEARCH with NEXT SENTENCE. Note, however, that if the NEXT SENTENCE phrase is executed, control will not pass to the next statement following the END-SEARCH, but instead will pass to the statement after the closest following period.

As an IBM extension, for the Format-2 SEARCH ALL statement, neither imperative-statement-2 nor NEXT SENTENCE is required. Without them, the SEARCH statement sets the index to the value in the table that 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 263.

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:

1. The condition(s) in the WHEN phrase are evaluated in the order in which they are written.
2. 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.
3. 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.
4. 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.

If indexing is used to search a table without an INDEXED BY clause, correct results are ensured only if both the table defined with the index and the table defined without the index have table elements of the same length and with the same number of occurrences.

identifier-2

Must be either an index data item or an elementary integer item. Identifier-2 cannot be a windowed date field. 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.

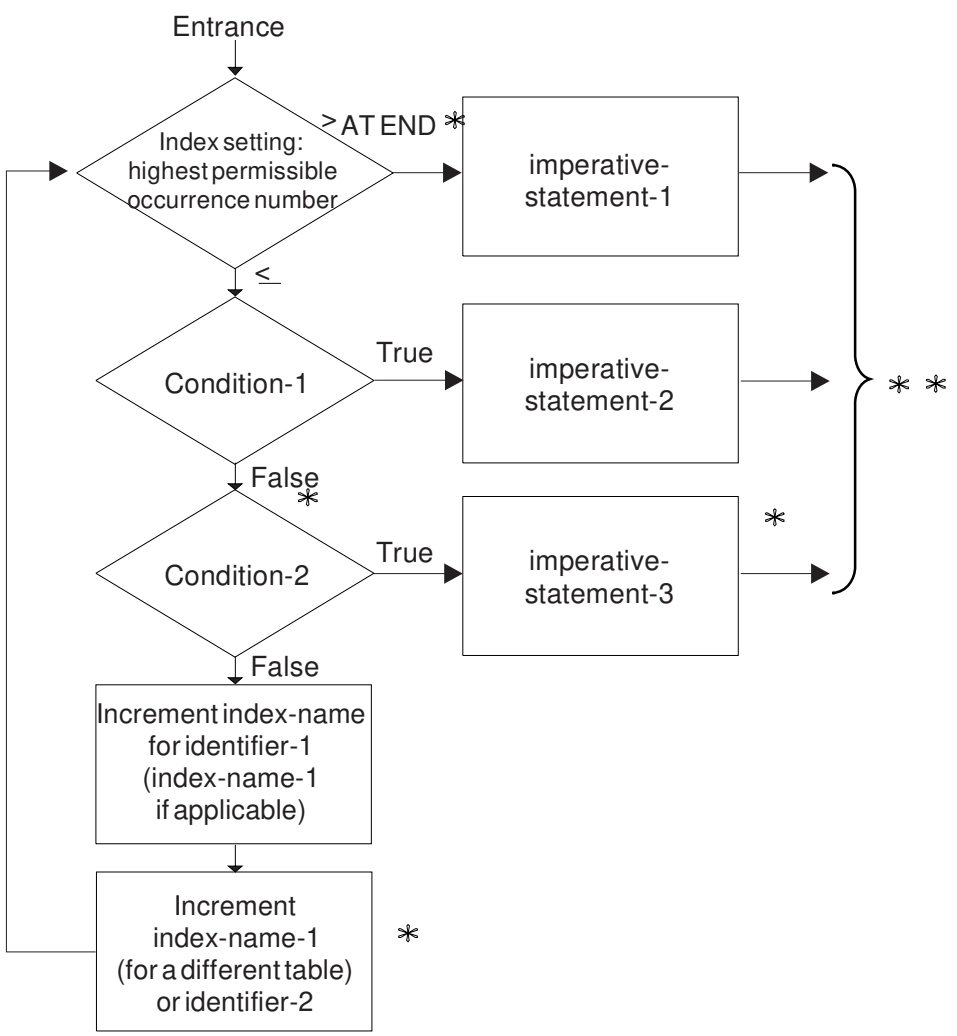
WHEN Phrase (Serial Search)

condition-1

Can be any condition described under “Conditional Expressions” on page 239.

Figure 17 illustrates a Format 1 SEARCH operation containing two WHEN phrases.

SEARCH Statement



* 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 order

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 2- to 7-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
A windowed date field

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-1
condition-name-2
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-1
data-name-2
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.

Data-name-1 and data-name-2 cannot be:

Floating-point data items

Group items containing variable occurrence data items

Windowed date fields

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.

Identifier-3 and identifier-4 can be floating-point data items.

Identifier-3 and identifier-4 cannot be data items defined with USAGE IS POINTER, USAGE IS PROCEDURE-POINTER, or USAGE IS OBJECT REFERENCE.

Identifier-3 and identifier-4 cannot be windowed date fields.

arithmetic-expression
Can be any of the expressions defined under “Arithmetic Expressions” on page 233, 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 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

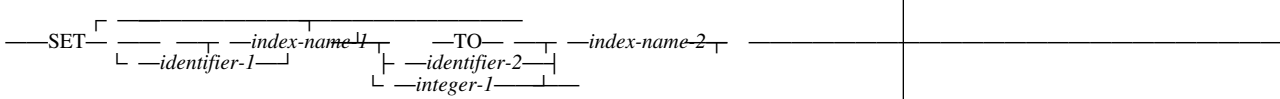
Index-names are related to a given table through the INDEXED BY phrase of the OCCURS clause; they are not further defined in the program.

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. The receiving fields cannot be windowed date fields.

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. The sending field cannot be a windowed date field.

integer-1

Sending field.

Must be a positive integer.

Table 48 shows valid combinations of sending and receiving fields in a Format 1 SET statement.

Table 48. 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.

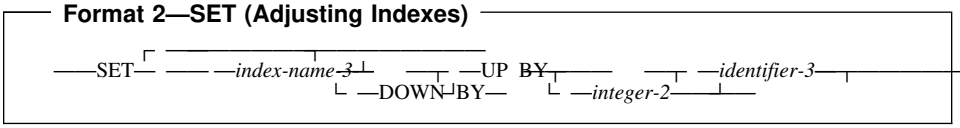
If index-name-2 is for a table that has a subordinate item that contains an OCCURS DEPENDING ON clause, then undefined values can be received into identifier-1.

For more information on complex OCCURS DEPENDING ON, see the *IBM COBOL 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. Identifier-3 cannot be a windowed date field.

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.

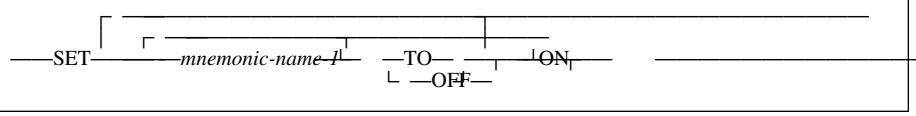
If index-name-3 is for a table that has a subordinate item that contains an OCCURS DEPENDING ON clause, and if the ODO object is changed before executing a Format 2 SET Statement, then index-name-3 cannot contain a value that corresponds to an occurrence number of its associated table.

For more information on complex OCCURS DEPENDING ON, see the *IBM COBOL 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)

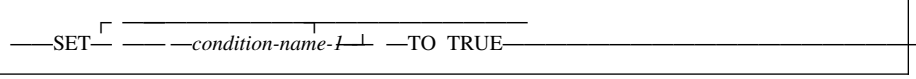


mnemonic-name
Must be associated with an external switch, the status of which can be altered.

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.

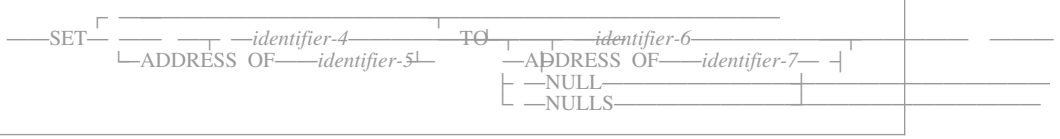
If more than one literal is specified in the VALUE clause of condition-name-1, its associated conditional variable is set equal to the first literal.

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.

Workstation 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. Workstation

Host For OS/390 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. Host

NULL
NULLS
Sending field.

Sets the receiving field to contain the value of an invalid address.

Table 49 shows valid combinations of sending and receiving fields in a Format 5 SET statement.

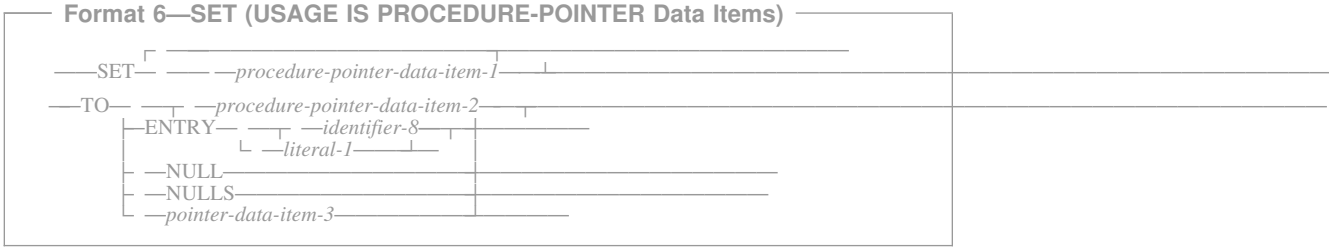
Table 49. Sending and Receiving Fields for Format 5 SET Statement

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 77. 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 77.

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 (OS/390)

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:

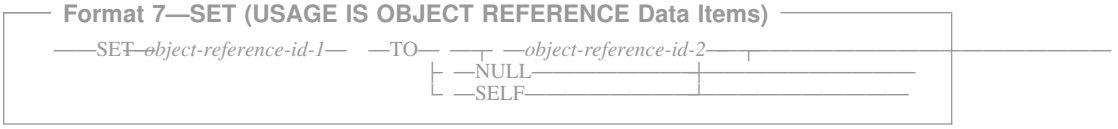
```
IDENTIFICATION DIVISION.
PROGRAM-ID DEMO.
DATA DIVISION.
WORKING-STORAGE SECTION.
 1 FP USAGE POINTER.
 1 PP USAGE PROCEDURE-POINTER.
PROCEDURE DIVISION.
      CALL "c-function" RETURNING FP.
      SET PP TO FP.
     CALL  PP.
```

For OS/390 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.

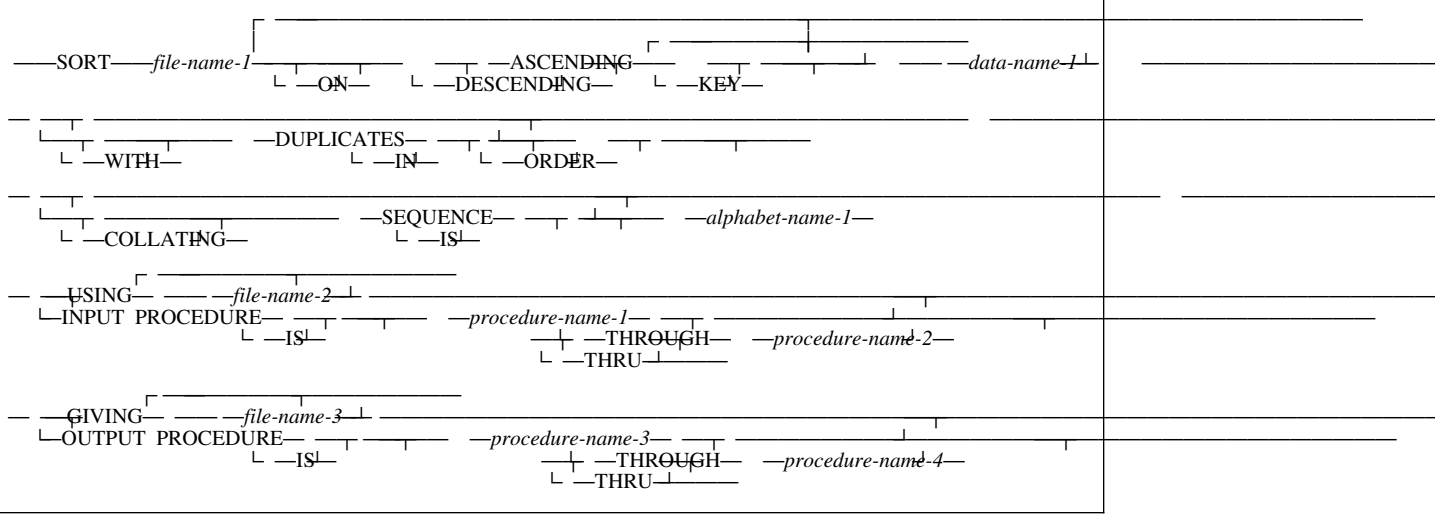
If the figurative constant NULL is specified, the receiving object-reference-id-1 is set to the NULL value.

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 345. 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.

File-names associated with the GIVING clause (file-name-3...) can be specified in the 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:

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.

KEY data items can be floating-point items.

KEY data items cannot be variably-located.

Workstation Under AIX, OS/2, and Windows, KEY data items cannot be windowed date fields. Workstation

Host Under OS/390 and VM, KEY data items can be windowed date fields, under these conditions:

- The GIVING phrase must not specify an indexed file, because the (binary) ordering assumed or imposed by the file system conflicts with the windowed date ordering provided in the sort output. Attempting to write the windowed date merge output to such an indexed file will either fail or reimpose binary ordering, depending on how the file is accessed (the ACCESS MODE in the file-control entry).
- If an alphanumeric windowed date field is specified as a KEY for a SORT statement, the collating sequence in effect for the merge operation must be EBCDIC. Thus the COLLATING SEQUENCE phrase of the SORT statement or, if this phrase is not specified, then any PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph, must not specify a collating sequence other than EBCDIC or NATIVE.

If the SORT statement meets these conditions, then the sort operation takes advantage of SORT Year 2000 features, assuming that the execution environment includes a sort product that supports century windowing.

A year-last windowed date field can be specified as a KEY for a SORT statement, and can thereby exploit the corresponding century windowing capability of the sort product.

For more information on using windowed date fields as KEY data items, see the *IBM COBOL Programming Guide* for your platform. Host

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 410).

Host Under OS/390 and VM, if the KEY is a DBCS item, the sequence of the KEY values is based on the binary collating sequence of the hexadecimal values of the DBCS characters. Host

Workstation Under AIX, OS/2, and Windows, if the KEY is a DBCS item, then the sequence of the KEY values is based on a collation sequence according to the COLLSEQ compiler option:

- If the COLLSEQ(NATIVE) compiler option is in effect, then the collating sequence is determined by the locale. For information on the locale, see Appendix F, "Locale Considerations (Workstation Only)" on page 568.
- Otherwise, the collating sequence is determined by the binary values of the DBCS characters.

Workstation

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 239).

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

Host Under OS/390 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 548.) Host

Workstation Under AIX, OS/2, and Windows the collating sequence is based on the character's hex value order. Workstation

STANDARD-2

Host Under OS/390 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. Host

Workstation Under AIX, OS/2, and Windows, the collating sequence is based on the character's hex value order. Workstation

NATIVE

Host Under OS/390 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 548.) Host

Workstation Under AIX, OS/2, and Windows, the collating sequence indicated by the locale is selected. Workstation

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 548.)

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.

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.

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 317).

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 18.

SORT-CORE-SIZE Special Register
See “SORT-CORE-SIZE” on page 17.

SORT-FILE-SIZE Special Register
See “SORT-FILE-SIZE” on page 17.

SORT-MODE-SIZE Special Register
See “SORT-MODE-SIZE” on page 18.

SORT-CONTROL Special Register
See “SORT-CONTROL” on page 16.

SORT-RETURN Special Register
See “SORT-RETURN” on page 18.

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:

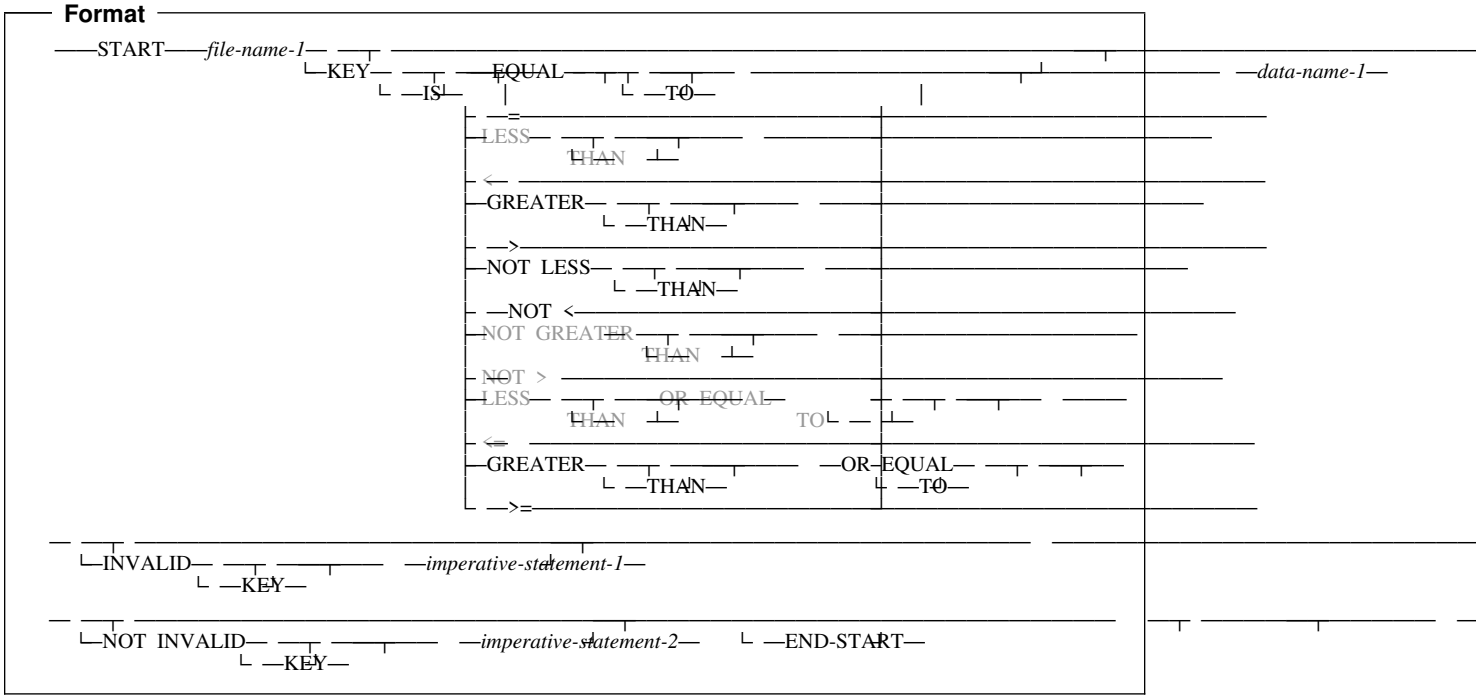
Totally within non-independent segments, or
Wholly within the same independent segment as that SORT statement.

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.

Workstation Under AIX, OS/2, and Windows, the following relational operators are allowed in the KEY phrase:

LESS THAN	<
NOT GREATER THAN	NOT >
LESS THAN OR EQUAL TO	<= Workstation

START Statement

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 270).

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 270.)

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 263.

Effect of Record Locking (OS/2 VSAM Files Only)

OS/2 For OS/2 VSAM files, 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. OS/2

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.

Data-name-1 need not be an alphanumeric item. However, for purposes of the I/O operation, 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

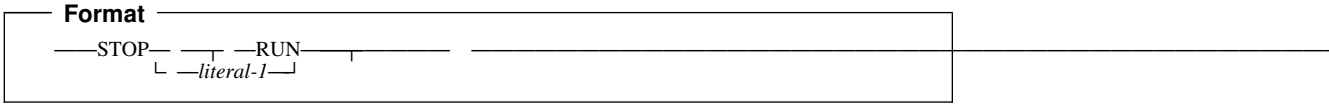
When the KEY phrase is specified, data-name-1 must specify the RELATIVE KEY.

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.

May be a signed numeric integer or non-integer literal, but may not be a floating-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.

Workstation Under AIX, OS/2, and Windows, do not use the STOP RUN or STOP literal statement in programs compiled with the THREAD compiler option. Workstation

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.

The STOP RUN statement does not have to be the last statement in a sequence, but the statements following the STOP RUN will not be executed.

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 application to end.)	Return directly to the program that called the main program. (May be the system and cause the application to end.)

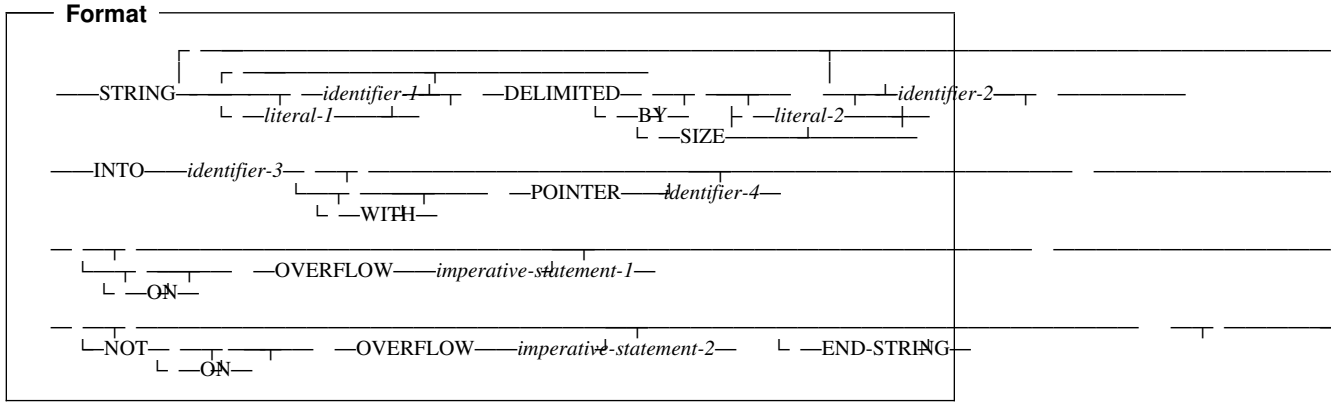
Host When running on OS/390 and using the linkage stack instructions, STOP RUN cannot be used to terminate a COBOL run unit from a subprogram if the linkage stack level has changed from the time the COBOL main program for that run unit was entered. Host

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



None of the identifiers in a STRING statement can be windowed date fields.

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 non-numeric literal.

identifier-1 through identifier-3
May be DBCS data items. If one of these identifiers is a DBCS item, then all of them, and all literals, must be DBCS items. May not be external floating-point items.

literal-1 and literal-2
May be DBCS literals. If one of these is a DBCS literal, then all of them must be DBCS literals, and identifier-1 through identifier-3 must be DBCS items.

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**.

It must not represent an edited data item or external floating-point item and must not be described with the JUSTIFIED clause. As an IBM extension, identifier-3 can 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.

When identifier-3 (the receiving field) is a DBCS data item, identifier-4 indicates the relative DBCS character position (not the relative byte position) in the receiving 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 263.

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 352).

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 424.

```
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

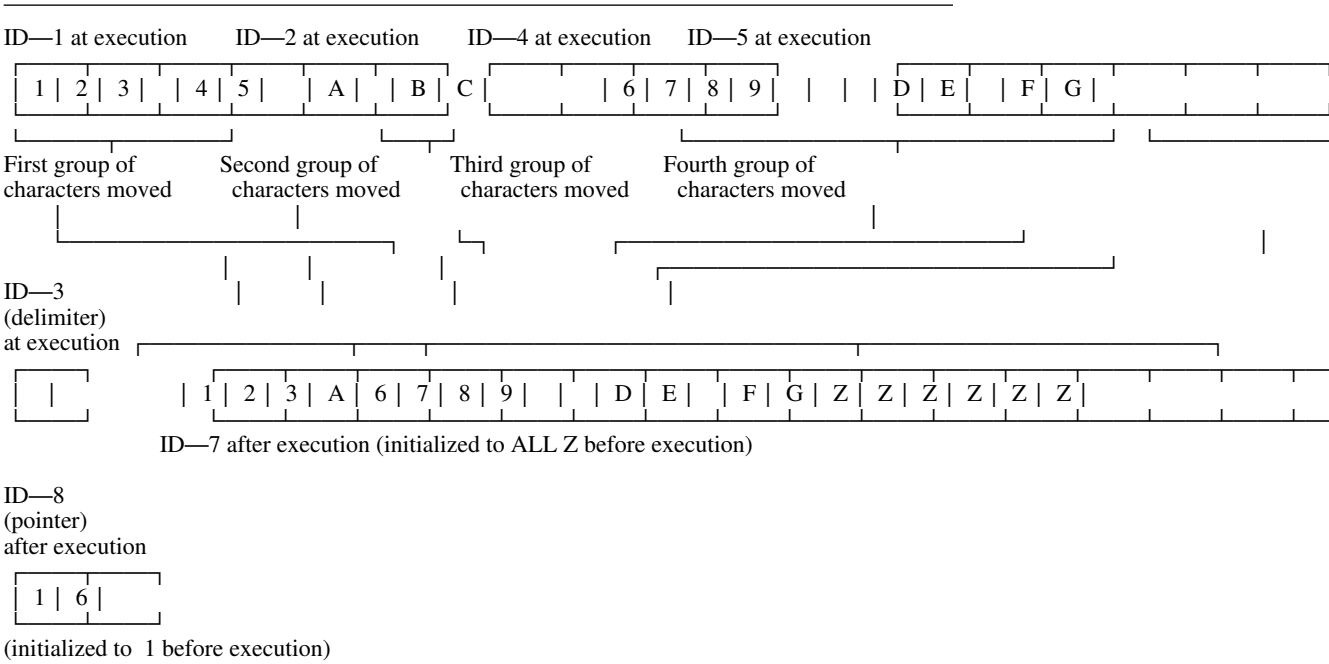
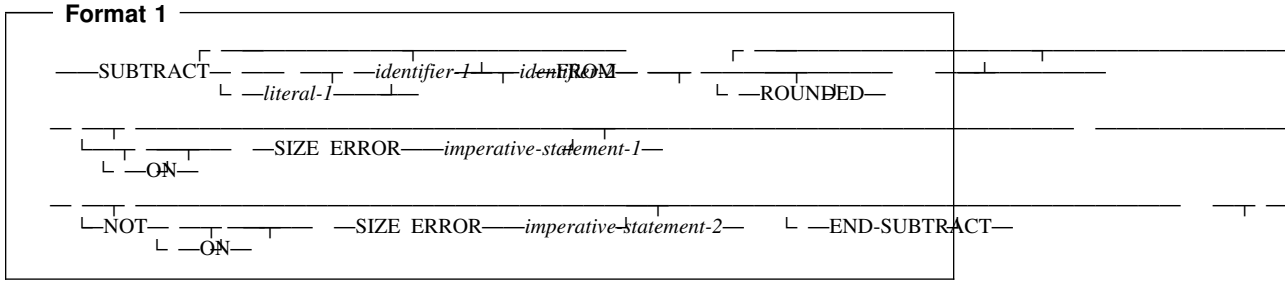


Figure 18. STRING Statement Execution Results

SUBTRACT Statement

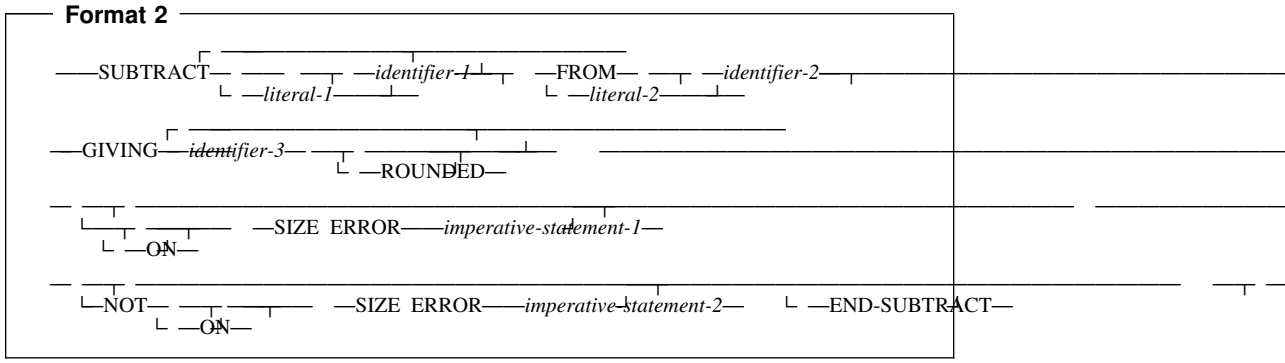
The SUBTRACT statement subtracts one numeric item, or the sum of two or more numeric items, from one or more numeric items, and stores the result.

Format 1



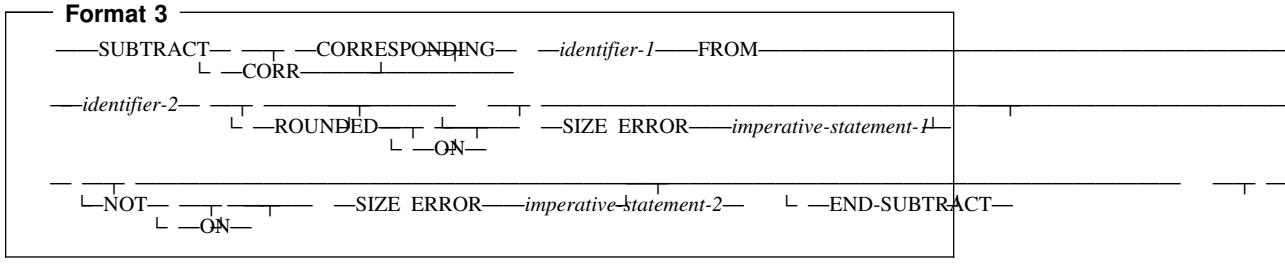
All identifiers or literals preceding the key word FROM are added together and this sum is subtracted from and stored immediately in identifier-2. This process is repeated for each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

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.

The composite of operands may contain more than 18 digits. For more information on arithmetic intermediate results, see the *IBM COBOL Programming Guide* for your platform.

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.

The following restrictions apply to date fields:

In Format 1, identifier-1 may specify at most one date field. If identifier-1 specifies a date field, then every instance of identifier-2 must specify a date field that is compatible with the date field specified by identifier-1. If identifier-1 does not specify a date field, then identifier-2 may specify one or more date fields, with no restriction on their DATE FORMAT clauses.

In Format 2, identifier-1 and identifier-2 may each specify at most one date field. If identifier-1 specifies a date field, then the FROM identifier-2 must be a date field that is compatible with the date field specified by identifier-1. Identifier-3 may specify one or more date fields. If identifier-2 specifies a date field and identifier-1 does not, then every instance of identifier-3 must specify a date field that is compatible with the date field specified by identifier-2.

In Format 3, if an item within identifier-1 is a date field, then the corresponding item within identifier-2 must be a compatible date field.

A year-last date field is allowed in a SUBTRACT statement only as identifier-1 and when the result of the subtraction is a non-date.

There are two steps to determining the result of a SUBTRACT statement that involves one or more date fields:

1. Subtraction: determine the result of the subtraction operation, as described under “Subtraction Involving Date Fields” on page 237.
2. Storage: determine how the result is stored in the receiving field. (In Formats 1 and 3, the receiving field is identifier-2; in Format 3, the receiving field is the GIVING identifier-3.) For details, see “Storing Arithmetic Results That Involve Date Fields” on page 237.

literal
Must be a numeric literal.

Floating-point data items and literals may be used anywhere numeric data items and literals can be specified.

ROUNDED Phrase

For information on the ROUNDED phrase, and for operand considerations, see “ROUNDED Phrase” on page 266.

SIZE ERROR Phrases

For information on the SIZE ERROR phrases, and for operand considerations, see “SIZE ERROR Phrases” on page 266.

CORRESPONDING Phrase (Format 3)

See “CORRESPONDING Phrase” on page 265.

END-SUBTRACT Phrase

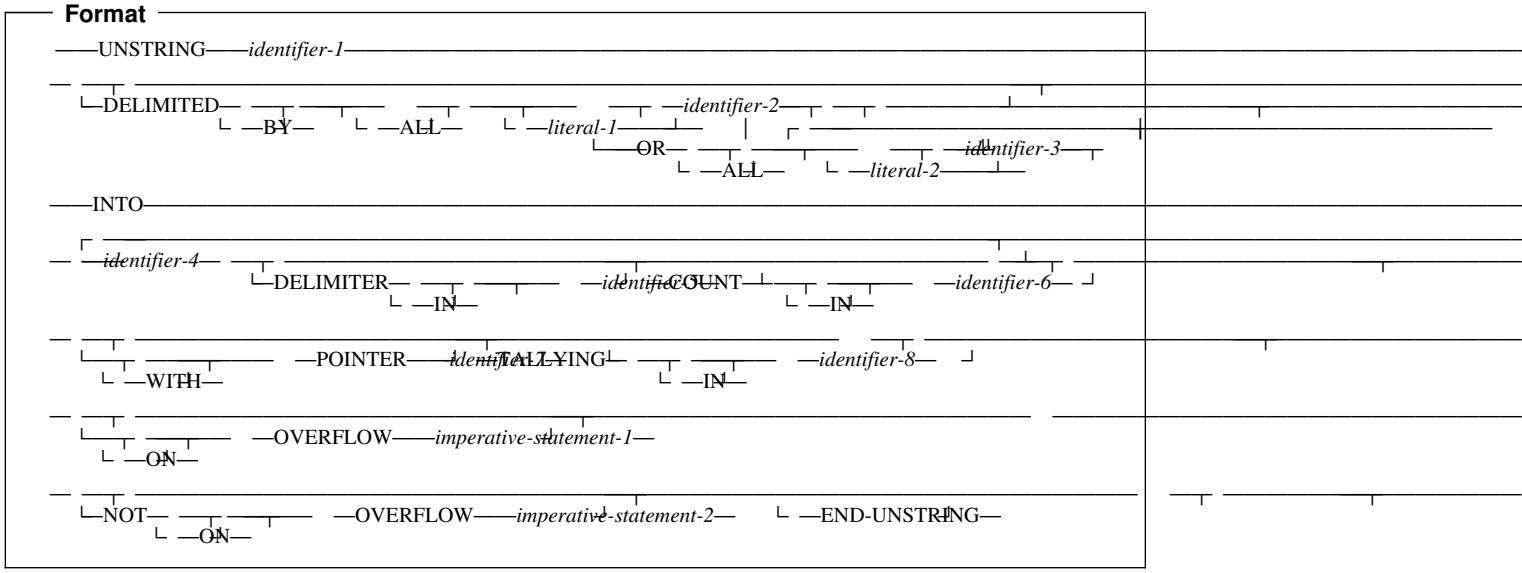
This explicit scope terminator serves to delimit the scope of the SUBTRACT statement. END-SUBTRACT permits a conditional SUBTRACT statement to be nested in another conditional statement. END-SUBTRACT may also be used with an imperative SUBTRACT statement.

For more information, see “Delimited Scope Statements” on page 263.

UNSTRING Statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

Format



None of the identifiers in an UNSTRING statement can be windowed date fields.

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.

As an IBM extension, identifier-1 can be reference-modified. It can be an alphanumeric-edited or an alphabetic data item. It can also be a 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 434.

The rules for moving an alphanumeric elementary item are the same as those for the MOVE statement (see “MOVE Statement” on page 352).

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.

When identifier-1 (the sending field) is a DBCS data item, identifier-7 indicates the 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 263.

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 41 on page 334.

Table 50. 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 352).
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.

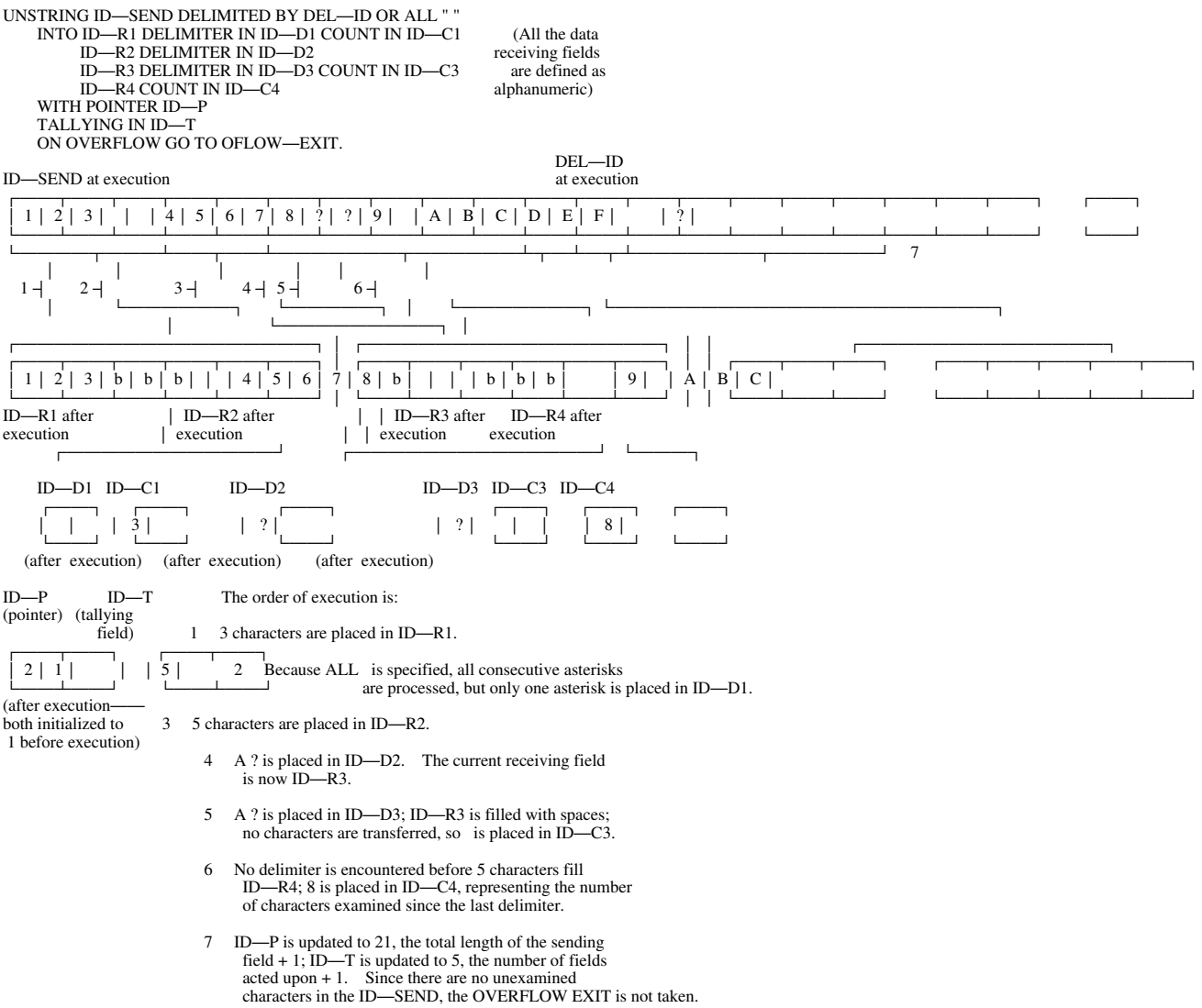


Figure 19. Results of UNSTRING Statement Execution

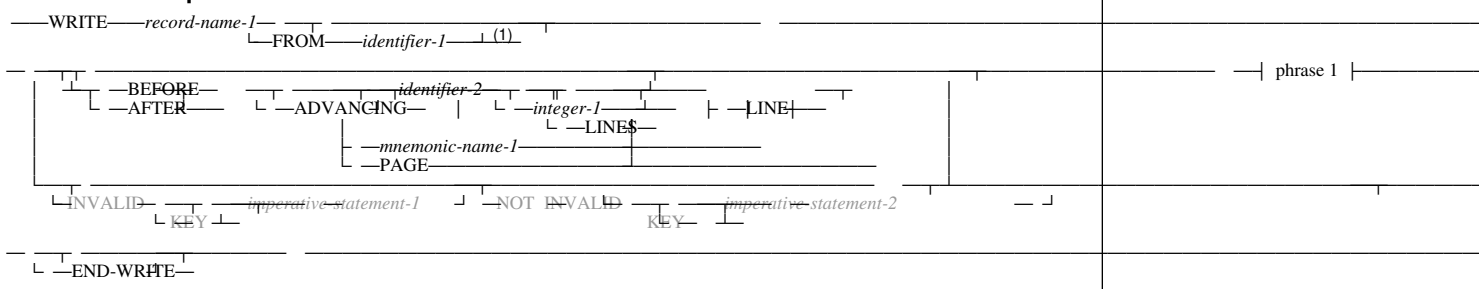
WRITE Statement

The WRITE statement releases a logical record for an output or input/output file.

When the WRITE statement is executed:

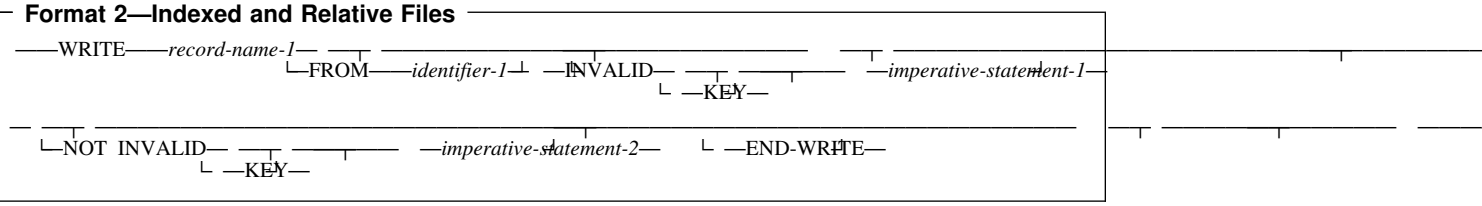
The associated sequential file must be open in OUTPUT or EXTEND mode.
The associated indexed or relative file must be open in OUTPUT, I-O, or EXTEND mode.

Format 1—Sequential Files

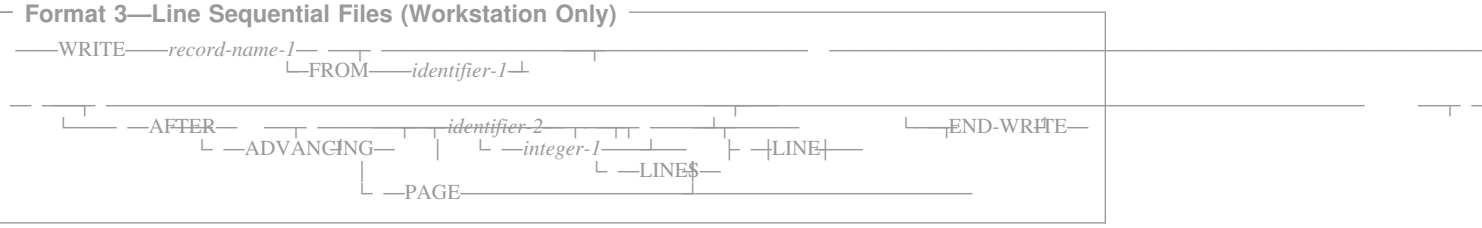


Note:
[1] The BEFORE, AFTER, INVALID KEY, and AT END OF PAGE phrases are not valid for STL files, OS/390 and VM VSAM files, or OS/2 Btrieve files.

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.

If record-name-1 is defined as a DBCS data item, Identifier-1 must be a DBCS data item.

For relative files, as an IBM extension, the number of character positions in the record-name can be different from the number of character positions in the record 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
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 270.)

identifier-2

Must be an integer data item.

ADVANCING Phrase

The ADVANCING phrase controls positioning of the output record on the page.

Workstation Under AIX, OS/2, and 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. Workstation

Host Under OS/390, the BEFORE and AFTER phrases are not supported for VSAM files. 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. Host

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. Identifier-2 cannot name a windowed date field.
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. Host 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.

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.
Host

When the ADVANCING phrase is omitted, automatic line advancing is provided, as if AFTER ADVANCING 1 LINE had been specified.

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.

Note: Host 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. Host

END-OF-PAGE Phrases

Workstation Under AIX, OS/2, and Windows, the END-OF-PAGE phrase is not supported for line sequential files or OS/2 Btrieve files. Workstation

Host Under OS/390, the AT END-OF-PAGE phrase is not supported for VSAM files. Host

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.

As an IBM extension, you can specify both the ADVANCING PAGE and END-OF-PAGE phrases in a single WRITE statement.

INVALID KEY Phrases

Workstation Under AIX, OS/2, and Windows, the INVALID KEY phrase is not supported for line sequential files or OS/2 Btrieve files. Workstation

Host Under OS/390, the INVALID KEY phrase is not supported for VSAM sequential files. Host

An invalid key condition is caused by the following:

For sequential files:

- 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 36 on page 271).

Otherwise, the WRITE statement is unsuccessful and the contents of record-name are unaffected (except for OS/390 QSAM files). And, the following occurs:

- **For sequential files**—the status key, if specified, is updated and an EXCEPTION/ERROR condition exists.

  If an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure is executed. If no such procedure is specified, the results are unpredictable.

- **For relative and indexed files**—program execution proceeds according to the rules described under "Invalid key condition" on page 272.

  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.

As an IBM extension, you can omit both the INVALID KEY phrase and the 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 263.

Effect of Record Locking (OS/2 VSAM Files Only)

OS/2 For OS/2 VSAM files, the following applies to files using record locking:

If two or more file connectors for a sequential file add records by sharing the file after opening it in EXTEND mode, the records are in an unspecified order.

If two or more file connectors for a relative file add records by sharing the file after opening it in EXTEND mode, the relative key values returned are ascending, but not necessary consecutive.

If two or more file connectors for an indexed file add records by sharing the file after opening it in EXTEND mode, the order of alternate keys allowing for duplicates is unspecified.

If the LOCK MODE IS AUTOMATIC is specified, a successful WRITE statement releases an existing record lock.

OS/2

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 190 and "USAGE Clause" on page 209.)

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

Host 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 Host

Punch Function Files with the IBM 3525

Host 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.

If stacker selection for the punch function file is desired, you can specify the appropriate stacker function-names in the SPECIAL-NAMES paragraph, and then issue WRITE ADVANCING statements using the associated mnemonic-names. Host

Print Function Files

Host 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 application, 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 51. Meanings of Environment-names in SPECIAL NAMES paragraph

Environment-name	Meaning
C02	**Line 3**
C03	**Line 5**
C04	**Line 7**
.	.
.	.
.	.
C12	**Line 23**

Host

Advanced Function Printing

Host 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/OS/390 (5695-040), PSF/VM (5684-141), or PSF/VSE (5686-040). Host

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 447
Specifying a Function 447
Function Definitions 454
ACOS 458
ANNUITY 459
ASIN 460
ATAN 461
CHAR 462
COS 463
CURRENT-DATE 464
DATE-OF-INTEGER 466
DATE-TO-YYYYMMDD 467
DATEVAL 468
DAY-OF-INTEGER 470
DAY-TO-YYYYDDD 471
FACTORIAL 472
INTEGER 473
INTEGER-OF-DATE 474
INTEGER-OF-DAY 475
INTEGER-PART 476
LENGTH 477
LOG 478
LOG10 479
LOWER-CASE 480
MAX 481
MEAN 482
MEDIAN 483
MIDRANGE 484
MIN 485
MOD 486
NUMVAL 487
NUMVAL-C 488
ORD 490
ORD-MAX 491
ORD-MIN 492
PRESENT-VALUE 493
RANDOM 494
RANGE 495
REM 496
REVERSE 497
SIN 498
SQRT 499
STANDARD-DEVIATION 500
SUM 501
TAN 502

© Copyright IBM Corp. 1991, 1998

UNDATE 503
UPPER-CASE 504
VARIANCE 505
WHEN-COMPILED 506
YEAR-TO-YYYY 508
YEARWINDOW 509

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.

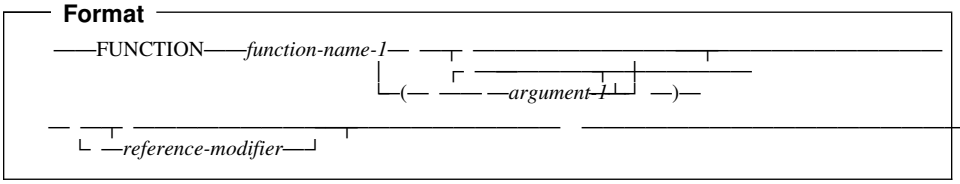
You can reference a function by specifying its name, along with any required arguments, in a Procedure Division statement.

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. Argument-1 cannot be a windowed date field, except in the UNDATE intrinsic function.

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.
```

© Copyright IBM Corp. 1991, 1998

replaces each lowercase letter in the argument with the corresponding uppercase letter, resulting in the movement of HELLO into DATA-NAME.

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.

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.

An argument cannot be a DBCS literal or data item. See "Function Definitions" on page 454 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.

Floating-point literals are allowed wherever a numeric argument is allowed, and in arithmetic expressions used in functions that allow a numeric argument. They are *not* allowed where an integer argument is required.

External floating-point items are allowed wherever a numeric argument is allowed, and in arithmetic expressions used in functions that allow a numeric argument.

External floating-point items are **not** allowed where an integer argument is required, or where an argument of alphanumeric class is allowed in a function identification, such as 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 run-time 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 argu-

ment 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:

```
1  PAYROLL.
 2  PAYROLL-WEEK      PIC  99.
  2 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 52 on page 455 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

The behavior of functions marked "DP" depends on whether the DATEPROC or NODATEPROC compiler option is in effect:

If the DATEPROC compiler option is in effect, the following intrinsic functions return date fields:

	Returned value has implicit DATE FORMAT...
DATE-OF-INTEGER	YYYYXXXX
DATE-TO-YYYYMMDD	YYYYXXXX
DAY-OF-INTEGER	YYYYXXX
DAY-TO-YYYYDDD	YYYYXXX
YEAR-TO-YYYY	YYYY
DATEVAL	Depends on the format specified by DATEVAL
YEARWINDOW	YYYY

If the NODATEPROC compiler option is in effect:

- The following intrinsic functions return the same values as when DATEPROC is in effect, but their returned values are non-dates:
  - DAY-OF-INTEGER
  - DATE-TO-YYYYMMDD
  - DAY-TO-YYYYDDD
  - YEAR-TO-YYYY
- The DATEVAL and UNDATE intrinsic functions have no effect, and simply return their (first) arguments unchanged
- The YEARWINDOW intrinsic function returns 0 unconditionally

Each intrinsic function is described in detail on the pages following the table.

Table 52 (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 DP	I1	I	Standard date equivalent (YYYYMMDD) of integer date
DATE-TO-YYYYMMDD DP	I1, I2	I	Standard date equivalent (YYYYMMDD) of I1 (standard date with a windowed year, YYMMDD), according to the 100-year interval whose ending year is specified by the sum of I2 and the year at execution time
DATEVAL DP	I1 or	I	Date field equivalent of I1 or X1
	X1	X	
DAY-OF-INTEGER DP	I1	I	Julian date equivalent (YYYYDDD) of integer date
DAY-TO-YYYYDDD DP	I1, I2	I	Julian date equivalent (YYYYDDD) of I1 (Julian date with a windowed year, YYDDD), according to the 100-year interval whose ending year is specified by the sum of I2 and the year at execution time
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

Table 52 (Page 2 of 3). Table of Functions

Function Name	Arguments	Type	Value Returned
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 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
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

Table 52 (Page 3 of 3). Table of Functions

Function Name	Arguments	Type	Value Returned
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
UNDATE DP	I1 or	I	Non-date equivalent of date field I1 or X1
	X1	X	
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
YEAR-TO-YYYY DP	I1, I2	I	Expanded year equivalent (YYYY) of I1 (windowed year, YY), according to the 100-year interval whose ending year is specified by the sum of I2 and the year at execution time
YEARWINDOW DP	None	I	If the DATEPROC compiler option is in effect, returns the starting year (in the format YYYY) of the century window specified by the YEARWINDOW compiler option; if NODATEPROC is in effect, returns 0

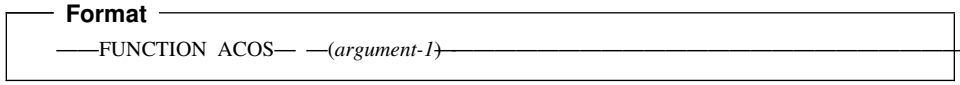
The following pages define each of the intrinsic functions summarized in the previous table.

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.

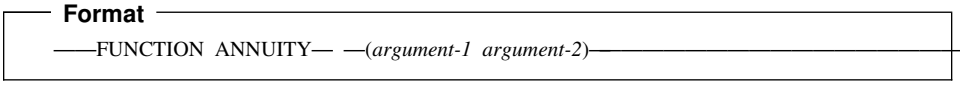
The returned value is the approximation of the arccosine of the argument and is greater than or equal to zero and less than or equal to Pi.

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:

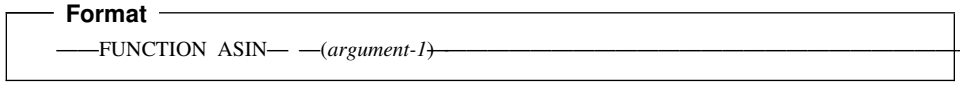
ARGUMENT-1 / (1 - (1 + ARGUMENT-1) (- ARGUMENT-2))

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.

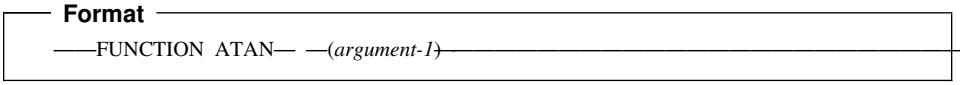
The returned value is the approximation of the arcsine of argument-1 and is greater than or equal to -Pi/2 and less than or equal to +Pi/2.

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.

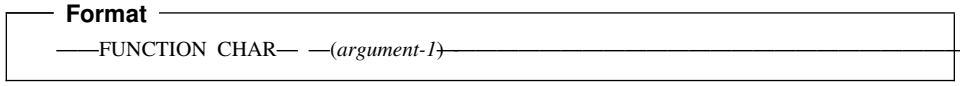
The returned value is the approximation of the arctangent of argument-1 and is greater than -Pi/2 and less than +Pi/2.

CHAR

The CHAR function returns a 1-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 OS/390 and VM, the EBCDIC collating sequence is used. (See Appendix B, “EBCDIC and ASCII Collating Sequences” on page 548.)

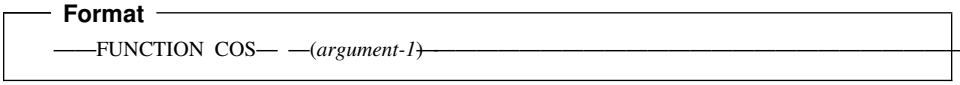
Under AIX, OS/2, and Windows, the COLLSEQ compiler option indicates the collating sequence used. For example, if COLLSEQ(EBCDIC) is specified and the PROGRAM COLLATING SEQUENCE is not specified (or is NATIVE), the EBCDIC collating sequence is applied.

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

```
FUNCTION CURRENT-DATE
```

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

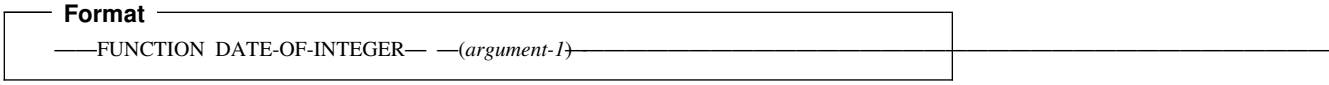
The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD).

The function type is integer.

The function result is an 8-digit integer.

If the DATEPROC compiler option is in effect, then the returned value is an expanded date field with implicit DATE FORMAT YYYYXXXX.

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.

Host Under OS/390 and VM, the INTDATE compiler option affects the starting date for the integer date functions. For details, see the *IBM COBOL for OS/390 & VM Programming Guide*. Host

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.

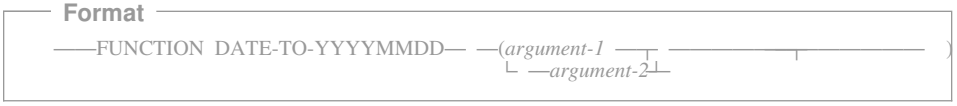
DATE-TO-YYYYMMDD

The DATE-TO-YYYYMMDD function converts argument-1 from a date with a 2-digit year (YYnnnn) to a date with a 4-digit year (YYYYnnnn). Argument-2, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding century window, into which the year of argument-1 falls.

The function type is integer.

If the DATEPROC compiler option is in effect, then the returned value is an expanded date field with implicit DATE FORMAT YYYYXXXX.

Format



argument-1
Must be zero or a positive integer less than 1,000,000.

argument-2
Must be an integer. If argument-2 is omitted, the function is evaluated assuming the value 50 was specified.

The sum of the year at the time of execution and the value of argument-2 must be less than 10,000 and greater than 1,699.

Example

Some examples of returned values from the DATE-TO-YYYYMMDD function follow:

Current Year	Argument-1 Value	Argument-2 Value	DATE-TO-YYYYMMDD Return Value
2002	851003	120	20851003
2002	851003	-20	18851003
2002	851003	10	19851003
1994	981002	-10	18981002

DATEVAL

The DATEVAL function converts a non-date to a date field, for unambiguous use with date fields.

If the DATEPROC compiler option is in effect, the returned value is a date field containing the value of argument-1 unchanged. For information on using the resulting date field:

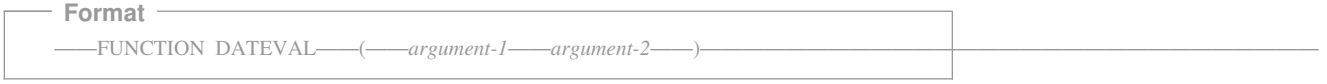
In arithmetic, see “Arithmetic with Date Fields” on page 235
In conditional expressions, see “Date Fields” on page 244

If the NODATEPROC compiler option is in effect, the DATEVAL function has no effect, and returns the value of argument-1 unchanged.

The function type depends on the type of argument-1:

Argument-1 Type	Function Type
Alphanumeric	Alphanumeric
Integer	Integer

Format



argument-1
Must be one of the following:

A class alphanumeric item with the same number of characters as the date format specified by argument-2.

An integer. This can be used to specify values outside the range specified by argument-2, including negative values.

The value of argument-1 represents a date of the form specified by argument-2.

argument-2

Must be a nonnumeric literal specifying a date pattern, as defined in "DATE FORMAT Clause" on page 164. The date pattern consists of YY or YYYY (representing a windowed year or expanded year, respectively), optionally preceded or followed by one or more Xs (representing other parts of a date, such as month and day), as follows. Note that the values are case-insensitive; the letters X and Y in argument-2 may be any mix of uppercase and lowercase.

Date-pattern string...	Specifies that argument-1 contains...
YY	A windowed (2-digit) year.
YYYY	An expanded (4-digit) year.
X	A single character; for example, a digit representing a semester or quarter (1–4).
XX	Two characters; for example, digits representing a month (01–12).
XXX	Three characters; for example, digits representing a day of the year (001–366).
XXXX	Four characters; for example, 2 digits representing a month (01–12) and 2 digits representing a day of the month (01–31).

Note: You can use apostrophes as the literal delimiters instead of quotes (independent of the APOST/QUOTE compiler option).

DAY-OF-INTEGER

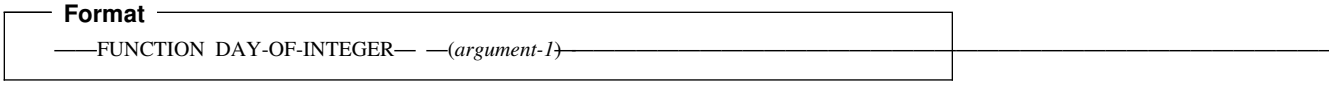
The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD).

The function type is integer.

The function result is a 7-digit integer.

If the DATEPROC compiler option is in effect, then the returned value is an expanded date field with implicit DATE FORMAT YYYYXXX.

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.

Host Under OS/390 and VM, the INTDATE compiler option affects the starting date for the integer date functions. For details, see the *IBM COBOL for OS/390 & VM Programming Guide*. Host

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.

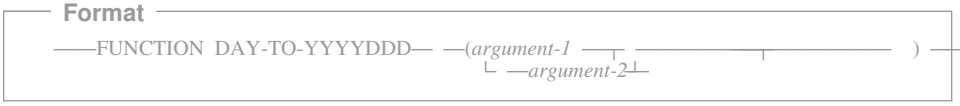
DAY-TO-YYYYDDD

The DAY-TO-YYYYDDD function converts argument-1 from a date with a 2-digit year (YYnnn) to a date with a 4-digit year (YYYYnnn). Argument-2, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding century window, into which the year of argument-1 falls.

The function type is integer.

If the DATEPROC compiler option is in effect, then the returned value is an expanded date field with implicit DATE FORMAT YYYYXXX.

Format



argument-1
Must be zero or a positive integer less than 100,000.

argument-2
Must be an integer. If argument-2 is omitted, the function is evaluated assuming the value 50 was specified.

The sum of the year at the time of execution and the value of argument-2 must be less than 10,000 and greater than 1,699.

Example

Some examples of returned values from the DAY-TO-YYYYDDD function follow:

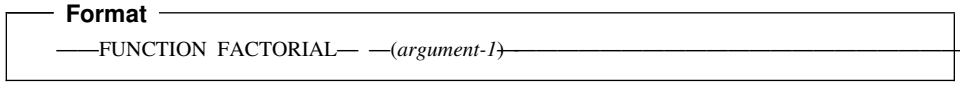
Current Year	Argument-1 Value	Argument-2 Value	DAY-TO-YYYYDDD Return Value
2002	10004	-20	1910004
2002	10004	-120	1810004
2002	10004	20	2010004
2013	95005	-10	1995005

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.

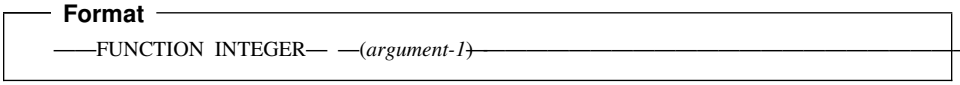
If the value of argument-1 is zero, the value 1 is returned; otherwise, its factorial is returned.

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.

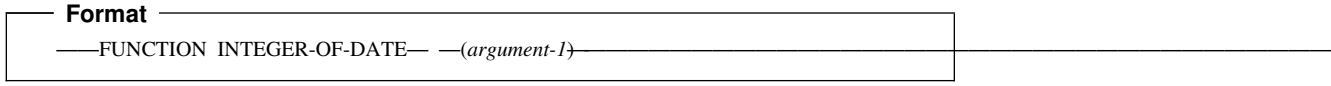
INTEGER-OF-DATE

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 7-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.

Host Under OS/390 and VM, the INTDATE compiler option affects the starting date for the integer date functions. For details, see the *IBM COBOL for OS/390 & VM Programming Guide*. Host

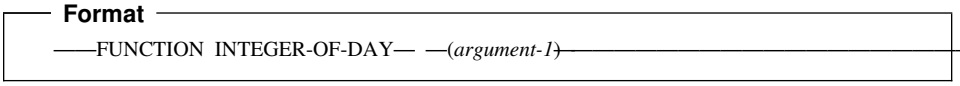
INTEGER-OF-DAY

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 7-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.

Host Under OS/390 and VM, the INTDATE compiler option affects the starting date for the integer date functions. For details, see the *IBM COBOL for OS/390 & VM Programming Guide*. Host

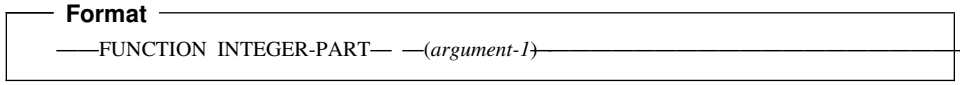
The returned value is an integer that is the number of days the date represented by argument-1, succeeds December 31, 1600 in the Gregorian calendar.

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.

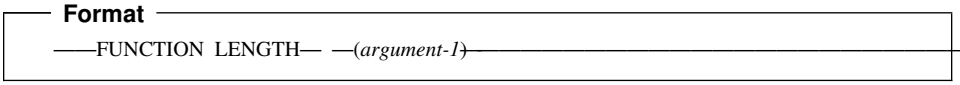
If the value of argument-1 is zero, the returned value is zero. If the value of argument-1 is positive, the returned value is the greatest integer less than or equal to the value of argument-1. If the value of argument-1 is negative, the returned value is the least integer greater than or equal to the 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 9-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.

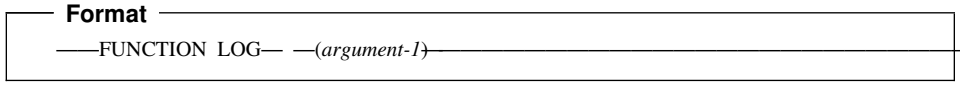
The returned value includes implicit FILLER characters, if any.

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.

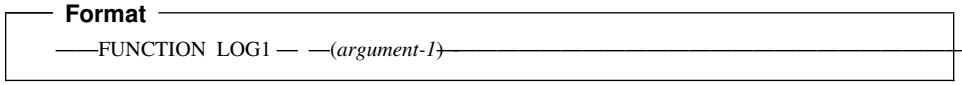
The returned value is the approximation of the logarithm to the base ***e*** of argument-1.

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.

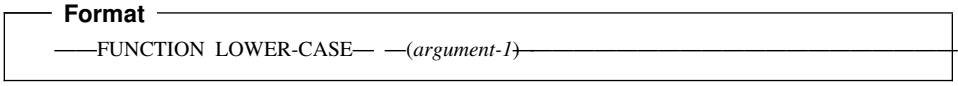
The returned value is the approximation of the logarithm to the base 10 of argument-1.

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.

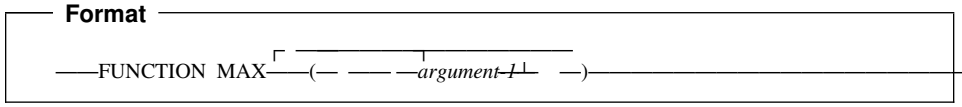
MAX

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.

If more than one argument-1 is specified, the combination of alphabetic and alphanumeric 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 239.

If more than one argument has the same greatest value, the leftmost argument having that value is returned.

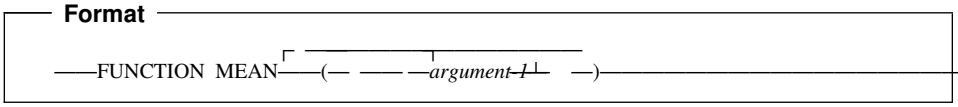
If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected argument.

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.

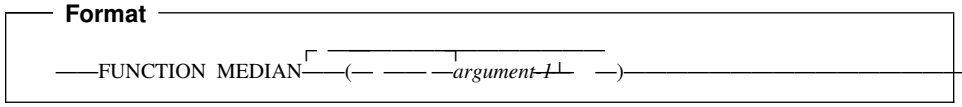
The returned value is the arithmetic mean of the argument-1 series. The returned value is defined as the sum of the argument-1 series divided by the number of occurrences referenced by argument-1.

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.

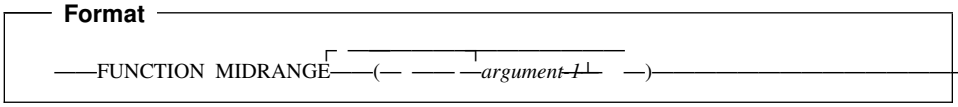
The comparisons used to arrange the argument values in sorted order are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 239.

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 239.

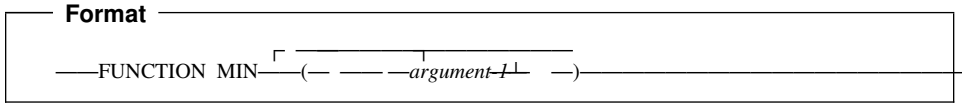
MIN

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.

If more than one argument-1 is specified, the combination of alphabetic and alphanumeric 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 239.

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

```
——FUNCTION MOD— —(argument-1 argument-2)——————————————————————————————
```

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 table illustrates the expected results for some values of argument-1 and argument-2.

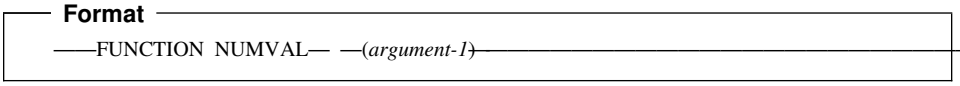
Argument-1	Argument-2	Return
11	5	1
-11	5	4
11	-5	-4
-11	-5	-1

NUMVAL

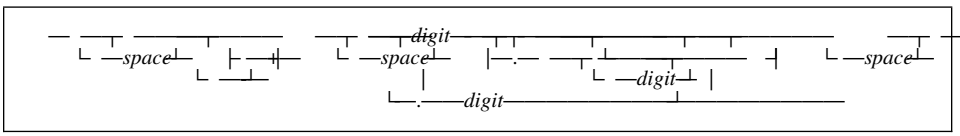
The NUMVAL function returns the numeric value represented by the alphanumeric character string specified in an argument. The function strips away any leading or trailing blanks in the string, producing a numeric value that can be used in an arithmetic expression.

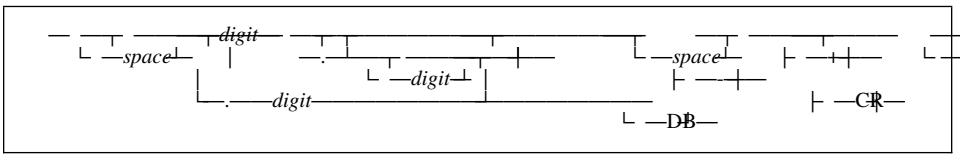
The function type is numeric.

Format



argument-1
must be a nonnumeric literal or an alphanumeric data item whose content has either of 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

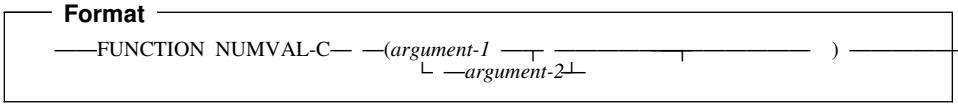
The NUMVAL-C function returns the numeric value represented by the alphanumeric character string specified as argument-1. Any optional currency sign specified by argument-2 and any optional commas preceding the decimal point are stripped away, producing a numeric value that can be used in an arithmetic expression.

The function type is numeric.

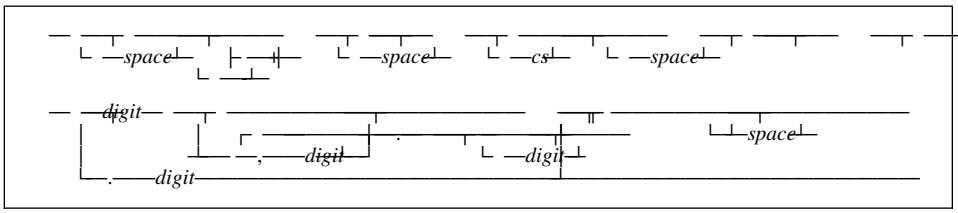
The NUMVAL-C function cannot be used if any of the following are true:

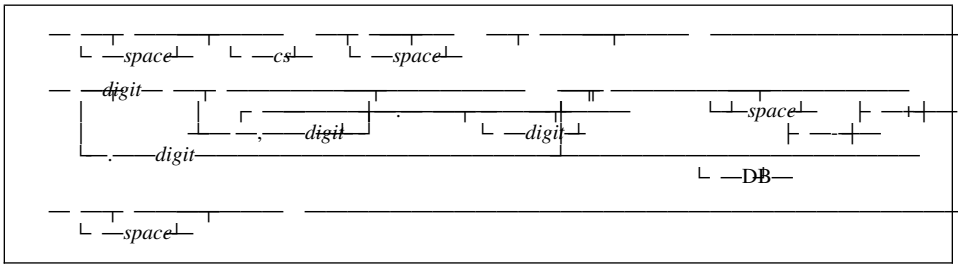
- The program contains more than one CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division.
- Literal-6 in the CURRENCY SIGN clause is a lowercase letter.
- The PICTURE SYMBOL paragraph is specified in the CURRENCY SIGN clause.

Format



argument-1
Must be a nonnumeric literal or an alphanumeric data item whose content has either of 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.

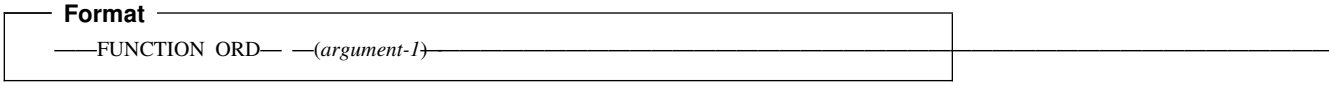
ORD

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 3-digit integer.

Format



argument-1
Must be one character in length and must be class alphabetic or alphanumeric.

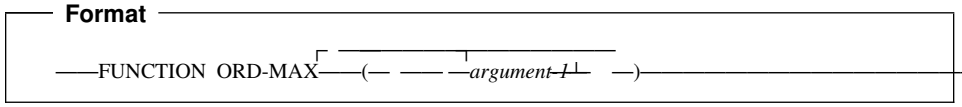
The returned value is the ordinal position of argument-1 in the collating sequence for the program; it ranges from 1 to 256 depending on the collating sequence.

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.

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 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 239.

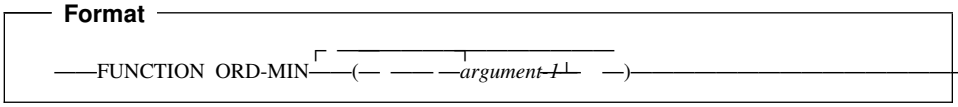
If more than one argument-1 has the same greatest value, the number returned corresponds to the position of the leftmost argument-1 having that value.

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 239.

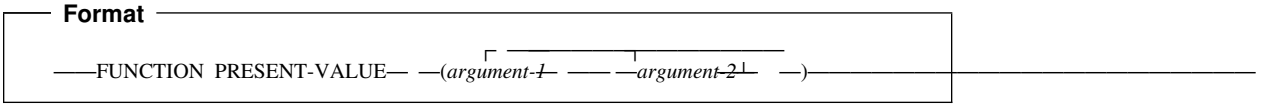
If more than one argument-1 has the same least value, the number returned corresponds to the position of the leftmost argument-1 having that value.

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

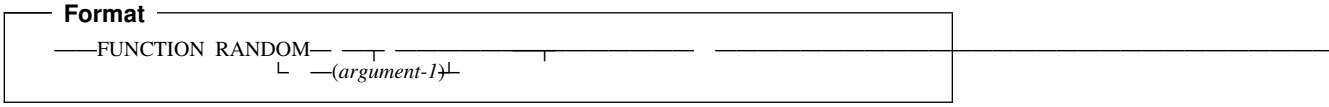
There is one term for each occurrence of argument-2. The exponent, n, is incremented from one by one for each term in the series.

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.

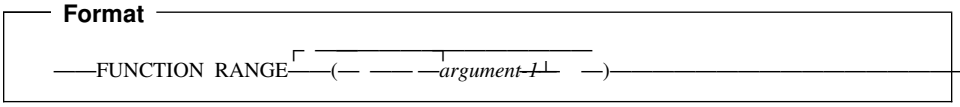
RANGE

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.

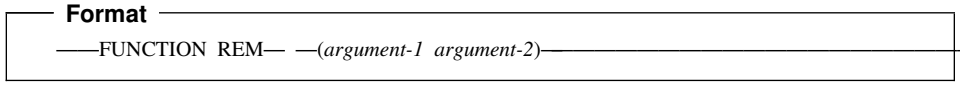
The returned value is equal to argument-1 with the greatest value minus the argument-1 with the least value. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 239.

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:

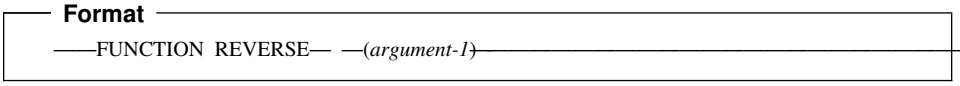
argument-1 - (argument-2 * FUNCTION INTEGER-PART (argument-1/argument-2))

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.

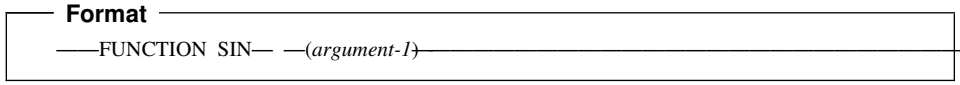
If argument-1 is a character string of length n, the returned value is a character string of length n such that, for 1 <= j <= n, the character in position j of the returned value is the character from position n-j+1 of argument-1.

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.

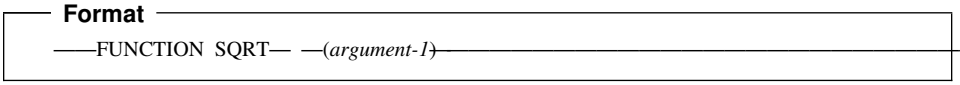
The returned value is the approximation of the sine of argument-1 and is greater than or equal to -1 and less than or equal to +1.

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.

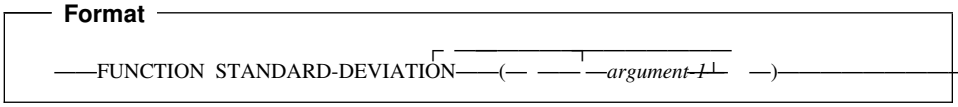
The returned value is the absolute value of the approximation of the square root of argument-1.

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.

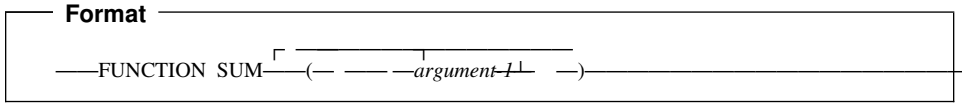
SUM

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.

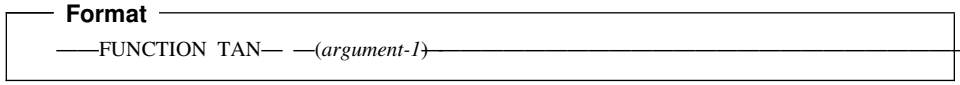
The returned value is the sum of the arguments. If the argument-1 series are all integers, the value returned is an integer. If the argument-1 series are not all integers, a numeric value is returned.

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.

UNDATE

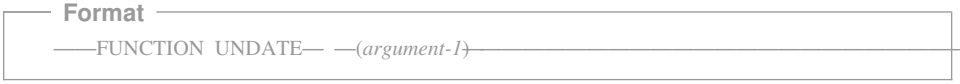
The UNDATE function converts a date field to a non-date for unambiguous use with non-dates.

If the NODATEPROC compiler option is in effect, the UNDATE function has no effect.

The function type depends on the type of argument-1:

Argument-1 Type	Function Type
Alphanumeric	Alphanumeric
Integer	Integer

Format



argument-1
A date field.

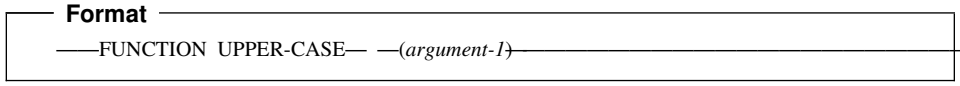
The returned value is a non-date that contains the value of argument-1 unchanged.

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.

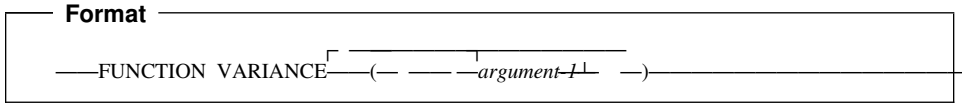
The character string returned has the same length as argument-1.

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

```
──FUNCTION WHEN-COMPILED──
```

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 containing program.

YEAR-TO-YYYY

The YEAR-TO-YYYY function converts argument-1, a 2-digit year, to a 4-digit year. Argument-2, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding century window, into which the year of argument-1 falls.

The function type is integer.

If the DATEPROC compiler option is in effect, then the returned value is an expanded date field with implicit DATE FORMAT YYYY.

Format

argument-1
Must be a non-negative integer that is less than 100.

argument-2
Must be an integer. If argument-2 is omitted, the function is evaluated assuming the value 50 was specified.

The sum of the year at the time of execution and the value of argument-2 must be less than 10,000 and greater than 1,699.

Example

Two examples of return values from the YEAR-TO-YYYY function follow:

Current Year	Argument-1 Value	Argument-2 Value	YEAR-TO-YYYY Return Value
1995	4	23	2004
1995	4	-15	1904
2008	98	23	1998
2008	98	-15	1898

YEARWINDOW

If the DATEPROC compiler option is in effect, the YEARWINDOW function returns the starting year of the century window specified by the YEARWINDOW compiler option. The returned value is an expanded date field with implicit DATE FORMAT YYYY.

If the NODATEPROC compiler option is in effect, the YEARWINDOW function returns 0.

The function type is integer.

Format

YEARWINDOW

Part 8. Compiler-Directing Statements

Compiler-Directing Statement 512
BASIS Statement 512
CBL (PROCESS) Statement 513
*CONTROL (*CBL) Statement 514
COPY Statement 516
DELETE Statement 523
EJECT Statement 524
ENTER Statement 524
INSERT Statement 525
READY or RESET TRACE Statement 526
REPLACE Statement 527
SERVICE LABEL Statement 530
SERVICE RELOAD Statement 531
SKIP1/2/3 Statements 531
TITLE Statement 532
USE Statement 533

Compiler Directives 539
CALLINTERFACE 539

© Copyright IBM Corp. 1991, 1998

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 516.

The source file remains unchanged after execution of the BASIS statement.

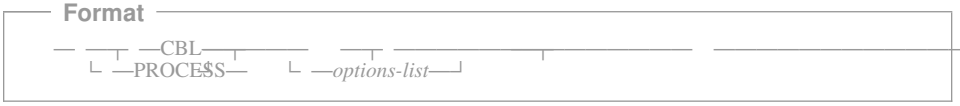
Note: If INSERT or DELETE statements are used to modify the COBOL source program provided by a BASIS statement, the sequence field of the COBOL source program must contain numeric sequence numbers in ascending order.

© Copyright IBM Corp. 1991, 1998

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.

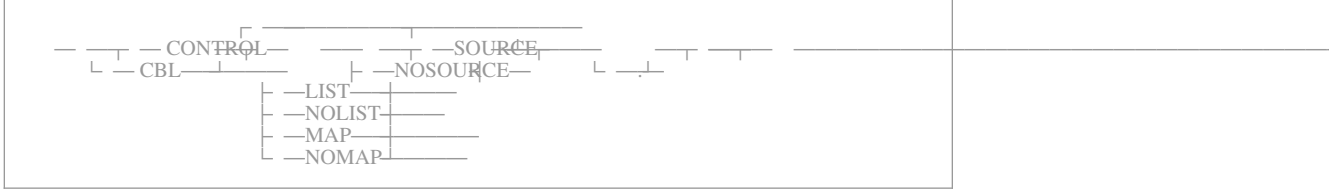
The CBL (PROCESS) statement must end before or at column 72, and options cannot be continued across multiple CBL (PROCESS) statements. However, you can use more than one CBL (PROCESS) statement. If you use multiple CBL (PROCESS) statements, they must follow one another with no intervening statements 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 applications, 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
    1  A                      1  A
    2  B                      2  B
CONTROL MAP                   CBL MAP
```

COPY Statement

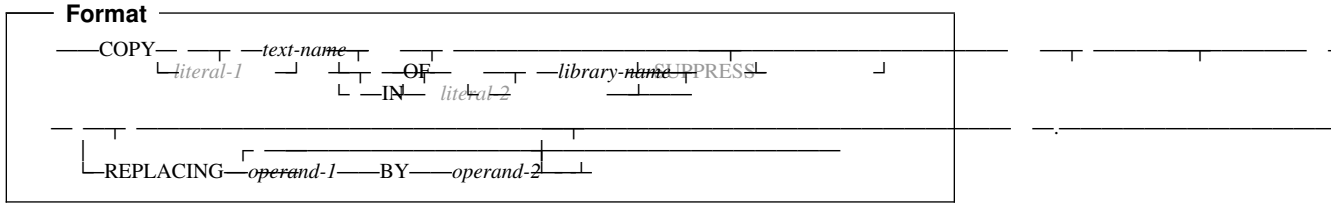
The COPY statement is a library statement that places prewritten text in a COBOL program.

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.
The word can contain characters: A-Z, a-z, 0-9, hyphen.
At least one character must be alphabetic.
The hyphen can't be the first or last character.

Host For OS/390 and VM, only the first eight characters are used as the identifying name. Host

literal-1, literal-2

Literal-1 identifies the name of the copy text. Literal-2 identifies where the copy text exists.

Host On OS/390 and VM:

The first eight characters are used as the identifying name.
The literal can contain characters: A-Z, a-z, 0-9, hyphen, @, #, $.
The leading character must be alphabetic. Host

Workstation On AIX, OS/2, and Windows:

The literal can be from 1-160 characters in length. Workstation

The uniqueness of text-name and library-name is determined after the formation and conversion rules for a system-dependent name have been applied.

As an IBM extension, a user-defined word can be the same as a text-name or a library-name.

As an IBM extension, if more than one COBOL library is available during compilation, text-name need not be qualified. If text-name is not qualified, a library-name of SYSLIB 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 separator can appear. As an IBM extension, COPY statements can be nested. However, nested COPY statements cannot contain the REPLACING phrase, and a COPY statement with the REPLACING phrase cannot contain nested COPY statements.

A COPY statement cannot cause recursion. That is, a COPY member can be named only once in a set of nested COPY statements until the end-of-file for that COPY member is reached. For example, assume that the source program contains the statement: `COPY X.` and library-text X contains the statement: `COPY Y.`.

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.

COPY Statement

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.

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 520).

Lines containing *CONTROL (*CBL), EJECT, SKIP1/2/3, or TITLE statements can occur in library text. Such lines are treated as comment lines during COPY statement 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.

DBCS words, DBCS literals, and EUC words are allowed in library text and pseudo-text.

SUPPRESS Phrase

The SUPPRESS phrase specifies that the library text is not to be printed on the source 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 2.

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.

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.

Pseudo-text can contain DBCS or EUC user-defined words, but the characters cannot be continued across lines.

identifier

Can be defined in any Data Division section.

literal

Can be numeric or nonnumeric.

Can be a DBCS literal.

word

Can be any single COBOL word (except COPY).

You can include the non-separator COBOL characters (for example, +, *, /, $, <, >, and =) as part of a COBOL word when used as REPLACING operands. In addition, 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.

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).

   When the library text contains a closing quotation mark that is not immediately followed by a separator space, comma, semicolon, or period, the closing quotation mark will be considered a separator quotation mark.
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.
13. COPY REPLACING does not affect the EJECT, SKIP1/2/3, or TITLE compiler-directing statements. When text words are placed in the source program, additional spaces are introduced only between text words where there already exists a 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:

```
1  A.
   2  B      PIC  S99.
   2  C      PIC  S9(5)V99.
   2  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:

```
1  A.
   2  B      PIC  S99.
   2  C      PIC  S9(5)V99.
   2  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:

```
 1  PAYROLL.
   2  PAY-CODE       PIC  S99.
   2  GROSS-PAY      PIC  S9(5)V99.
   2  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:

```
 1  :TAG:.
   2  :TAG:-WEEK              PIC  S99.
   2  :TAG:-GROSS-PAY         PIC  S9(5)V99.
   2  :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:

```
 1  PAYROLL.
   2  PAYROLL-WEEK            PIC  S99.
   2  PAYROLL-GROSS-PAY       PIC  S9(5)V99.
   2  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 ==( 1)== BY ==( 1)==
                   ==  1 == BY ==  5 ==.
```

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:

```
25 DELETE     1 -  5 ,  4 ,  45
```

Source program statements can follow a DELETE statement. These source program statements are then inserted into the BASIS source program before the statement fol-

ENTER Statement

lowing 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.

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 applications, 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.

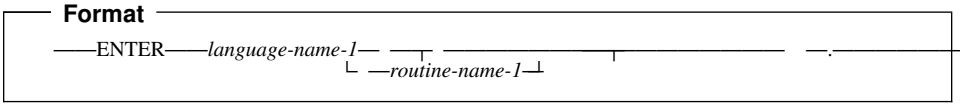
ENTER Statement

The ENTER statement allows the use of more than one source language in the same source program.

With COBOL for OS/390 & VM, COBOL Set for AIX, and VisualAge COBOL, 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-39  WITH DEBUGGING MODE.
             .
DATA  DIVISION.
             .
  WORKING-STORAGE  SECTION.
   1 TRACE-SWITCH            PIC 9 VALUE  .
     88   READY-TRACE              VALUE  1.
     88   RESET-TRACE             VALUE   .
             .
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

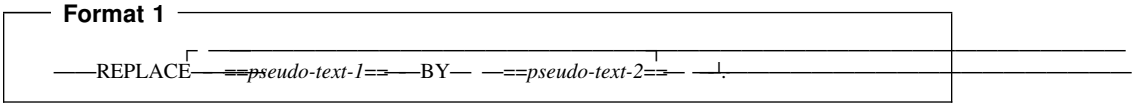
The REPLACE statement is used to replace source program text.

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.

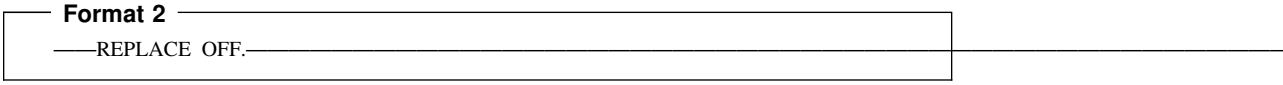
If the word REPLACE appears in a comment-entry or in the place where a comment-entry can appear, it is considered part of the comment-entry.

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.

As an IBM extension, pseudo-text-1 can consist entirely of a separator comma or a 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.

REPLACE Statement

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.

The REPLACE statement can be used with DBCS literals and DBCS or EUC names.

Pseudo-text can contain DBCS or EUC character-strings, but the characters cannot be 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 35.)

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.

However, when pseudo-text-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).

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.

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 32). 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.

Lines containing *CONTROL (*CBL), EJECT, SKIP1/2/3, or TITLE statements can occur in source program text. Such lines are treated as comment lines during 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

Workstation Under AIX, OS/2, and Windows, the SERVICE LABEL statement is treated as a comment. Workstation

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 OS/390 & VM Programming Guide* for more information.

SERVICE RELOAD Statement

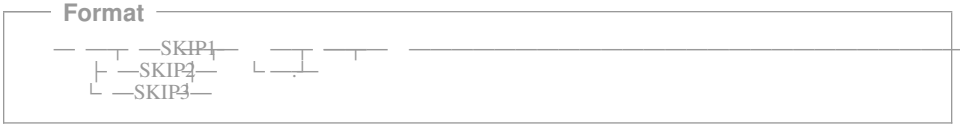
The SERVICE RELOAD statement is treated as a comment.

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 applications, 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 applications, 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
LABEL declarative (OS/390 and VM Only)
DEBUGGING declarative

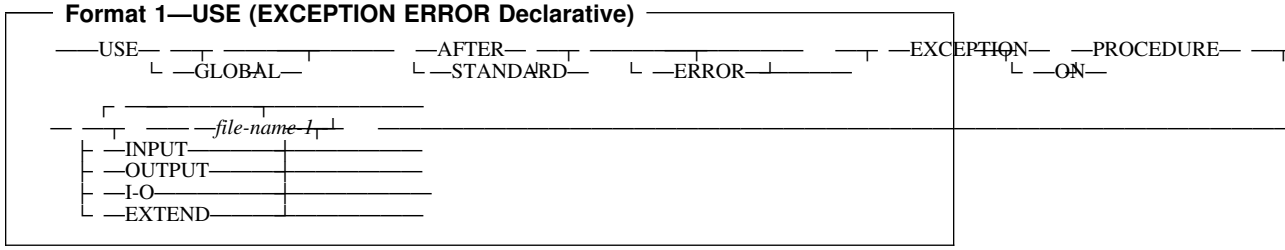
For general information on declaratives, see “Declaratives” on page 230.

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 270.)

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 270.

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, OS/390, and VM:

A declarative procedure can be performed from a nondeclarative procedure.

Additionally for OS/390 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.

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.

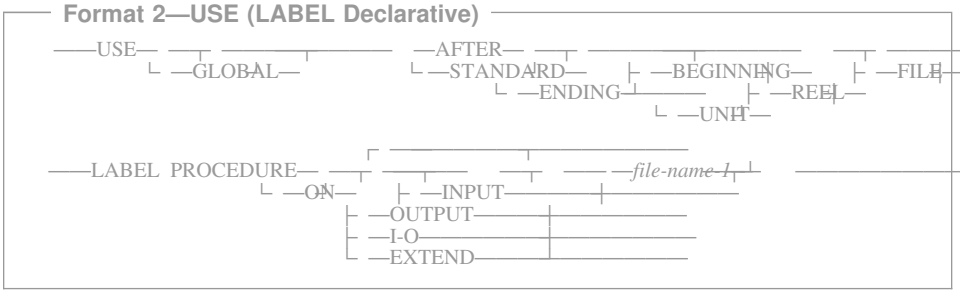
Steps 3. and 4. are repeated until the last examined program is the outermost program, or until a qualifying declarative has been found.

LABEL Declarative

The LABEL declarative provides user label-handling procedures.

Workstation Under AIX, OS/2, and Windows, USE...AFTER...LABEL PROCEDURE is not supported. If encountered, they are ignored and a warning message is issued.
Workstation

Format 2—USE (LABEL Declarative)



USE Statement

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)

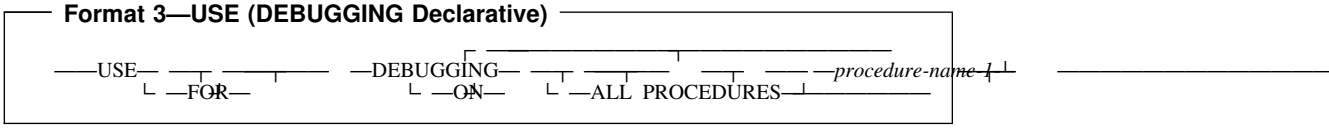
The WITH DEBUGGING MODE clause of the SOURCE compiler statement activates all debugging sections and lines that have been compiled into the object program. See Appendix C, “Source Language Debugging” on page 555, for additional details.

USE Statement

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 53 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 53. 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.

Host Under OS/390 and VM, the CALLINTERFACE directive is not supported. Host

Format

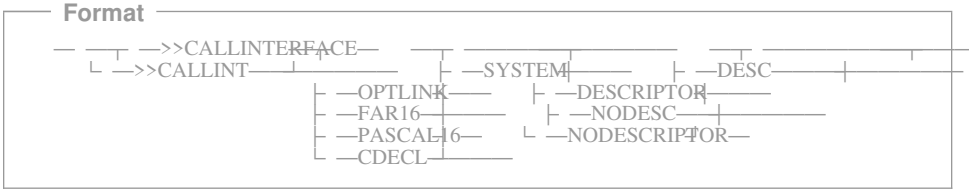


Table 54. CALLINTERFACE Options supported by Platform

AIX	OS/2	Windows
SYSTEM	SYSTEM OPTLINK FAR16 PASCAL16	SYSTEM OPTLINK CDECL
DESC NODESC	DESC NODESC	DESC 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 541.

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, 1998

CALLINTERFACE

FAR16
Specifies that the _FAR16_CDecl calling convention as defined by C Set++ is used 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.

Specify CALLINTERFACE only in the Procedure Division.

The positions of CALL statements relative to the >>CALLINTERFACE directive are recognized following any processing of COPY and REPLACE statements. For example, CALL statements and >>CALLINTERFACE statements in COPY text are processed by the rules specified for the directive.

Syntax and General Rules

You must specify a >>CALLINTERFACE on a line by itself, in Area B.

You cannot specify >>CALLINTERFACE:

- Within a source text manipulation sentence (for example, COPY or REPLACING)
- Between the lines of a continued character string
- On a debugging line
- In the middle of a COBOL statement

The >>CALLINTERFACE specification is limited to the current program.

The REPLACE statement and REPLACE phrase of the COPY statement do not affect the CALLINTERFACE specification.

Difference Between the Directive and Compiler Option

You can indicate which calling convention you want by using either the CALLINTERFACE directive or the CALLINT compiler option. Use the directive when you want to use more than one call convention for the CALL statements in a compilation unit. Use the compiler option when you want to use the same call convention for the entire compilation unit.

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, 1998

Appendix A. Compiler Limits

The following table lists the compiler limits for IBM COBOL programs running under OS/390, VM, AIX, OS/2, and Windows.

Table 55 (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)	No limit
Number of COPY libraries	No limit
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 ...	No limit
literal THRU/ALSO ...	256
Input-Output Section	
FILE-CONTROL paragraph	
SELECT file-name ...	65,535
ASSIGN system-name ...	No limit[2]
ALTERNATE RECORD KEY data-name ...	253
RECORD KEY length	No limit[3]
RESERVE integer (buffers)	255[4]
I-O-CONTROL paragraph	
RERUN ON system-name ...	32,767
integer RECORDS	16,777,215
SAME RECORD AREA	255
FOR file-name ...	255
SAME SORT/MERGE AREA	No limit[2]

© Copyright IBM Corp. 1991, 1998

Table 55 (Page 2 of 4). Compiler Limits

Language Element	Compiler Limit
MULTIPLE FILE ... file-name	No limit[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 ...	No limit[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 ...	No limit[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 ...	No limit
VALUE literal ...	No limit
66 RENAMES ...	No limit
PICTURE character-string	30
Numeric item digit positions	18
Numeric-edited 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
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

Compiler Limits

Table 55 (Page 3 of 4). Compiler Limits

Language Element	Compiler Limit
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)	No limit
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 ...	No limit
ALTER pn1 TO pn2 ...	4,194,303[1]
CALL ... BY CONTENT id	2,147,483,647 bytes
CALL id/lit USING id/lit...	16380 (OS/390 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)	No limit
CANCEL id/lit ...	No limit
CLOSE file-name ...	No limit
COMPUTE identifier ...	No limit
DISPLAY id/lit ...	No limit
DIVIDE identifier ...	No limit
ENTRY USING id/lit ...	No limit
EVALUATE ... subjects	64
EVALUATE ... WHEN clauses	256
GO pn ... DEPENDING	255
INSPECT TALLY/REPL clauses	No limit
MERGE file-name ASC/DES KEY ...	No limit
Total key length	4,092 bytes[6]
USING file-name ...	16[7]

Table 55 (Page 4 of 4). Compiler Limits

Language Element	Compiler Limit
MOVE id/lit TO id ...	No limit
MULTIPLY identifier ...	No limit
OPEN file-name	No limit
PERFORM	4,194,303
SEARCH ... WHEN ...	No limit
SET index/id ... TO	No limit
SET index ... UP/DOWN	No limit
SORT file-name ASC/DES KEY	No limit
Total key length	4,092 bytes[6]
USING file-name ...	16[7]
STRING identifier ...	No limit
DELIMITED id/lit ...	No limit
UNSTRING DELIMITED id/lit OR id/lit ...	255
UNSTRING INTO id/lit ...	No limit
USE ... ON file-name ...	No limit

Table Notes (OS/390 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 56 (Page 1 of 4). EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
65		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, 1998

Table 56 (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 56 (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 56 (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

US English ASCII Code Page (ISO 646)

Table 57 (Page 1 of 4). ASCII Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
1		Null	0	0
⋮				
33		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

ASCII Code Values

Table 57 (Page 2 of 4). ASCII Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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
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

Table 57 (Page 3 of 4). ASCII Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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
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

ASCII Code Values

Table 57 (Page 4 of 4). ASCII Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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 533. 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, 1998

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.

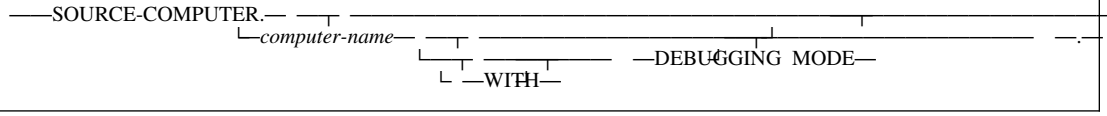
DEBUG-ITEM Special Register

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 OS/390 and VM

COBOL Set for AIX Programming Guide for AIX

VisualAge COBOL 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.

Source Language Debugging

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 COBOL for OS/390 & VM, COBOL Set for AIX, and VisualAge COBOL 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 COBOL for OS/390 & VM, COBOL Set for AIX, and VisualAge COBOL products. These reserved words include both reserved words for IBM extensions and a subset of the COBOL 85 Standard reserved words.

Reserved words marked 1 are applicable only to AIX, OS/2, and Windows.

Words marked under **Standard Only** are COBOL 85 Standard reserved words for function not implemented in 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.

Words marked X2 under **RFD** are reserved for future development under OS/390 and VM only.

Note: You can change which reserved word table is used 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 58 (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		
AREA	X		
AREAS	X		

Table 58 (Page 1 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
ARITHMETIC			X
ASCENDING	X		
ASSIGN	X		
AT	X		
AUTHOR	X		
AUTOMATIC 1	X		
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		
BOOLEAN			X
BOTTOM	X		
BY	X		
CALL	X		

© Copyright IBM Corp. 1991, 1998

Table 58 (Page 2 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
CANCEL	X		
CBL	X		
CD		X	
CF		X	
CH		X	
CHARACTER	X		
CHARACTERS	X		
CLASS	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 [1]	X		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 [1]	X		X[2]
COMPUTATIONAL-6			X
COMPUTATIONAL-7			X
COMPUTATIONAL-8			X
COMPUTATIONAL-9			X
COMPUTE	X		
CONFIGURATION	X		
CONNECT			X
CONTAINED			X

Table 58 (Page 2 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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		
DELIMITER	X		
DEPENDING	X		
DESCENDING	X		
DESTINATION		X	

Reserved Words

Table 58 (Page 3 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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		
END-RECEIVE		X	
END-RETURN	X		
END-REWRITE	X		
END-SEARCH	X		

Table 58 (Page 3 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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
FROM	X		
FUNCTION	X		
GENERATE		X	
GET			X

Table 58 (Page 4 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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	
INHERITS	X		
INITIAL	X		
INITIALIZE	X		
INITIATE		X	
INPUT	X		
INPUT-OUTPUT	X		
INSERT	X		
INSPECT	X		
INSTALLATION	X		
INTO	X		
INVALID	X		
INVOKE	X		
IS	X		
JUST	X		
JUSTIFIED	X		
KANJI	X		
KEEP			X
KEY	X		

Table 58 (Page 4 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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
LOCAL-STORAGE	X		
LOCK	X		
LOW-VALUE	X		
LOW-VALUES	X		
MEMBER			X
MEMORY	X		
MERGE	X		
MESSAGE		X	
METACLASS	X		
METHOD	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		
NULLS	X		
NUMBER		X	
NUMERIC	X		
NUMERIC-EDITED	X		

Reserved Words

Table 58 (Page 5 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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 [1]	X		X[2]
PRINTING		X	
PRIOR			X
PROCEDURE	X		
PROCEDURE-POINTER	X		
PROCEDURES	X		
PROCEED	X		
PROCESSING	X		
PROGRAM	X		
PROGRAM-ID	X		
PROTECTED			X

Table 58 (Page 5 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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		
RETURNING	X		
REVERSED	X		
REWIND	X		
REWRITE	X		

Table 58 (Page 6 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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		
SPACE	X		
SPACES	X		
SPECIAL-NAMES	X		
STANDARD	X		

Table 58 (Page 6 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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		
TRANSCEIVE			X
TRUE	X		
TYPE		X	
UNEQUAL			X

Reserved Words

Table 58 (Page 7 of 7). Reserved Words

Reserved Word	IBM COBOL	Standard Only	RFD
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 OS/390 and VM only.

Appendix E. ASCII Considerations for OS/390 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).

When nonnumeric comparisons within the object program are to use the ASCII collating sequence, the PROGRAM COLLATING SEQUENCE clause of the

© Copyright IBM Corp. 1991, 1998

ASCII Considerations

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-39
    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.

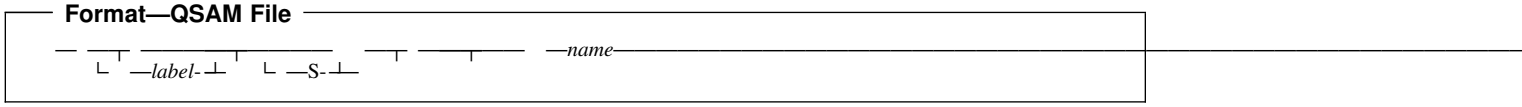
The PROGRAM COLLATING SEQUENCE clause, in conjunction with the alphabet-name clause, can be used to specify EBCDIC nonnumeric comparisons for an ASCII-encoded tape file or ASCII nonnumeric comparisons for an EBCDIC-encoded tape file.

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 specifies 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.

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 workstation products running on AIX, OS/2, and Windows, 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 and 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 workstation products running on AIX, OS/2, and Windows, 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 workstation products running on AIX, OS/2, and Windows, 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 workstation products running on AIX, OS/2, and Windows, 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, 1998

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

Table 59 lists the language elements that are different between COBOL for OS/390 & VM and the IBM Workstation COBOL compilers (COBOL Set for AIX and VisualAge COBOL). Many COBOL for OS/390 & 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 COBOL for OS/390 & VM language syntax, even if the language element has no functional effect.

Table 59 (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 OS/390 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 OS/390 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	Line sequential I-O is not supported on OS/390 and VM.
LOCK MODE clause	For OS/2 VSAM files, 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, 1998

Table 59 (Page 2 of 2). Language Difference Between Mainframe and Workstation IBM COBOL

Language Element	Implementation
POINTER and PROCEDURE-POINTER data items	Under COBOL for OS/390 & VM, a POINTER data item is defined as 4 bytes; a PROCEDURE-POINTER data item is defined as 8 bytes. Under AIX, OS/2, and Windows, 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	The contents of this special register differ between host and workstation COBOL.
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 IBM COBOL. Next is the 3-character abbreviation of the module name as used in the standard. Finally, the 2 digits separated by a comma indicate the minimum and maximum levels of the module. For example, (2 SEQ 1,2) means that IBM COBOL 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, 1998

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. Host FIPS Publication 21-4, Federal Information Processing Standard 21-4, COBOL high subset. Host
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*.

6. Workstation SPIRIT (Service Provider's Requirements for Information Technology), Part 6—COBOL Language Profile, published by Network Management Forum. Workstation
7. Workstation MIA (Multivendor Integration Architecture), technical requirements, specified by Nippon Telegraph and Telephone Corp (NTT). Workstation

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 OS/390 & VM

Fact Sheet, GC26-9048

Licensed Program Specifications, GC26-9044

Compiler and Run-Time Migration Guide, GC26-4764

Installation and Customization under OS/390, GC26-9045

Programming Guide, SC26-9049

Language Reference, SC26-9046

Debug Tool User's Guide and Reference, SC09-2137

Diagnosis Guide, GC26-9047

IBM COBOL Millennium Language Extensions Guide, GC26-9266

Related Publications for OS/390 and VM

IBM VisualAge COBOL Millennium Language Extensions for OS/390 & VM

Fact Sheet, GC26-9321

Licensed Program Specifications, GC26-9307

IBM COBOL Millennium Language Extensions Guide, GC26-9266

Installation and Customization under OS/390, GC26-9045

Language Environment

Concepts Guide, GC28-1945

OS/390 Customization, SC28-1941

Programming Guide, SC28-1939

Programming Reference, SC28-1940

Writing Interlanguage Applications, SC28-1943

Run-Time Migration Guide, SC28-1944

Debugging Guide and Run-Time Messages, SC28-1942

OS/390 SOMobjects™

Configuration and Administration Guide, GC28-1851

Getting Started, GA22-7248

Messages, Codes, and Diagnosis, SC28-1996

Object Services, SC28-1995

Programmer's Guide, GC28-1859

Programmer's Reference Volume 1, SC28-1997

Programmer's Reference Volume 2, SC28-1998

Programmer's Reference Volume 3, SC28-1999

Reference Summary, SC28-1856

OS/390

JCL Reference, GC28-1757

JCL User's Guide, GC28-1758

System Codes, GC28-1780

System Commands, GC28-1781

VM/ESA®

Application Development Guide, SC24-5450

Application Development Reference, SC24-5451

Command Reference, SC24-5461

User's Guide, SC24-5460

CP Command and Utility Reference, SC24-5519

System Messages and Codes, SC24-5529

XEDIT Command and Macro Reference, SC24-5464

XEDIT User's Guide, SC24-5463

CICS/ESA®

Application Programming Guide, SC33-1169

Application Programming Reference, SC33-1170

Sample Applications Guide, SC33-1173

DB/2

Application Programming and SQL Guide, SC26-3266

IMS/ESA®

Application Programming and SQL Guide, SC26-3266

DFSORT™

DFSORT Application Guide, SC33-4035

© Copyright IBM Corp. 1991, 1998

IBM COBOL Set for AIX

Fact Sheet, GC26-8484

Language Reference, SC26-9046

Programming Guide, SC26-8423

Program Builder User's Guide, SC09-2201

LPEX User's Guide and Reference, SC09-2202

Getting Started, GC26-8425

IBM COBOL Millennium Language Extensions Guide, GC26-9266

SMARTdata UTILITIES for AIX

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

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 COBOL

Fact Sheet, GC26-9052

Getting Started on OS/2, GC26-9051

Getting Started on Windows, GC26-8944

Language Reference, SC26-9046

Programming Guide, SC26-9050

Visual Builder User's Guide, SC26-9053

IBM COBOL Millennium Language Extensions Guide, GC26-9266

SMARTdata UTILITIES for OS/2

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

SMARTdata UTILITIES for Windows

SdU for Windows Distributed FileManager User's Guide, SC26-7134

Data Description and Conversion, SC26-7091

Data Description and Conversion A Data Language Reference, SC26-7092

VSAM Application Programming Interface Reference, SC26-7133

Related Publications for OS/2

DB2

Application Programming Guide, S20H-4643

SQL Reference, S20H-4665

CICS for OS/2

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

CICS for Windows NT

Installation Guide, SC33-1880

Application Programming Guide, SC33-1888

Intercommunication Guide, SC33-1882

Messages & Codes, SC33-1886

Problem Determination Guide, SC33-1883

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

Softcopy Publications for IBM COBOL

The following collection kits contain IBM COBOL or related product publications in BookManager® format:

MVS Collection, SK2T-0710

OS/390 Collection, SK2T-6700

VM Collection, SK2T-2067

To view the latest IBM COBOL product publications in Acrobat PDF format, visit the IBM COBOL Family home page on the World Wide Web at:

http://www.software.ibm.com/ad/cobol/

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 2-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, 1998

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. A century window is a 100-year interval within which any 2-digit year is unique. There are several types of century window available to COBOL programmers:

1. For windowed date fields, it is specified by the YEARWINDOW compiler option
2. For windowing intrinsic functions DATE-TO-YYYYMMDD, DAY-TO-YYYYDDD, and YEAR-TO-YYYY, it is specified by argument-2
3. For Language Environment callable services, it is specified in CEESCEN

*** 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
	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.

compatible date field. The meaning of the term "compatible," when applied to date fields, depends on the COBOL division in which the usage occurs:

Data Division
Two date fields are compatible if they have identical USAGE and meet at least one of the following conditions:

- They have the same date format.
- Both are windowed date fields, where one consists only of a windowed year, DATE FORMAT YY.
- Both are expanded date fields, where one consists only of an expanded year, DATE FORMAT YYYY.
- One has DATE FORMAT YYXXXX, the other, YYXX.
- One has DATE FORMAT YYYYXXXX, the other, YYYYXX.

A windowed date field can be subordinate to an expanded date group data item. The two date fields are compatible if the subordinate date field has USAGE DISPLAY, starts two bytes after the start of the group expanded date field, and the two fields meet at least one of the following conditions:

- The subordinate date field has a DATE FORMAT pattern with the same number of Xs as the DATE FORMAT pattern of the group date field.
- The subordinate date field has DATE FORMAT YY.
- The group date field has DATE FORMAT YYYYXXXX and the subordinate date field has DATE FORMAT YYXX.

Procedure Division
Two date fields are compatible if they have the same date format except for the year part, which may be windowed or expanded. For example, a windowed date field with DATE FORMAT YYXXX is compatible with:

- Another windowed date field with DATE FORMAT YYXXX
- An expanded date field with DATE FORMAT YYYYXXX

*** 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 value. A character-string that identifies the monetary units stored in a numeric-edited item. Typical examples are ‘$’, ‘USD’, and ‘EUR’. A currency sign value can be defined by either the CURRENCY compiler option or the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. If the CURRENCY SIGN clause is not specified and the NOCURRENCY compiler option is in effect, the dollar sign ($) is used as the default currency sign value. See also “currency symbol.”

currency symbol. A character used in a PICTURE clause to indicate the position of a *currency sign value* in a numeric-edited item. A currency symbol can be

defined by either the CURRENCY compiler option or by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. If the CURRENCY SIGN clause is not specified and the NOCURRENCY compiler option is in effect, the dollar sign ($) is used as the default currency sign value and currency symbol. Multiple currency symbols and currency sign values can be defined. See also "currency sign value."

*** 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.

date field. Any of the following:

A data item whose data description entry includes a DATE FORMAT clause.

A value returned by one of the following intrinsic functions:

DATE-OF-INTEGER
DATE-TO-YYYYMMDD
DATEVAL
DAY-OF-INTEGER
DAY-TO-YYYYDDD
YEAR-TO-YYYY
YEARWINDOW

The conceptual data items DATE, DATE YYYYMMDD, DAY, and DAY YYYYDDD of the ACCEPT statement.

The result of certain arithmetic operations (for details, see "Arithmetic with Date Fields" on page 235).

The term date field refers to both "expanded date field" and "windowed date field." See also "non-date."

date format. The date pattern of a date field, specified either:

Explicitly, by the DATE FORMAT clause or DATEVAL intrinsic function argument-2

or

Implicitly, by statements and intrinsic functions that return date fields (for details, see "Date Field" on page 59)

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 composed 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 2-character combination belonging to the following set:

Character	Meaning
	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 OS/390 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.”

expanded date field. A date field containing an expanded (4-digit) year. See also “date field” and “expanded year.”

expanded year. A date field that consists only of a 4-digit year. Its value includes the century: for example, 1998. Compare with “windowed year.”

*** 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 OS/390 & VM bootstrap routine. It must be link-edited with any module that contains a COBOL for OS/390 & 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 2-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 (such as COBOL for MVS & VM, COBOL for OS/390 & VM, 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 2-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.

locale. 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*.

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.

non-date. Any of the following:

A data item whose date description entry does not include the DATE FORMAT clause

A literal

A date field that has been converted using the UNDATE function

A reference-modified date field

The result of certain arithmetic operations that may include date field operands; for example, the difference between two compatible date fields

*** 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
 1  TABLE-1.
     5  X                          PICS9.
     5  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.

*** 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 1- or 2-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
	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

windowed date field. A date field containing a windowed (2-digit) year. See also “date field” and “windowed year.”

windowed year. A date field that consists only of a 2-digit year. This 2-digit year may be interpreted using a century window. For example, 05 could be interpreted as 2005. See also “century window.”

Compare with “expanded year.”

*** 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.

Y

Z

zoned decimal item. See “external decimal item.”

Index

Special Characters

, (comma)
  insertion character 190
  symbol in PICTURE clause 181, 185
/ (slash)
  insertion character 190
  symbol in PICTURE clause 181, 185
(/) comment line 38
 (period) 182
<= (less than or equal to) 244
< (less than) 244
{ : }
  description 29
  required use of 522
$ (default currency symbol)
  in PICTURE clause 182, 185
  insertion character 191, 192
* symbol in PICTURE clause 182
*CBL (*CONTROL) statement 514
+ (plus)
  insertion character 191, 192, 193
  SIGN clause 201
  symbol in PICTURE clause 185
– (minus)
  insertion character 191, 192
  SIGN clause 201
  symbol in PICTURE clause 185
= (equal) 244
> (greater than) 244
>= (greater than or equal to) 244

Numerics

0
  insertion character 190
  symbol in PICTURE clause 181, 185
66, RENAMES data description entry 198
77, item description entry 140
88, condition-name data description entry 162
9, symbol in PICTURE clause 181, 185

A

A, symbol in PICTURE clause 179
abbreviated combined relation condition
  examples 260
  using parentheses in 259
ACCEPT statement
  mnemonic name in 277, 278
  overlapping operands, unpredictable results 269
  system information transfer 279
  under AIX and OS/2 277
access mode
  description 115
  dynamic
    DELETE statement 304
    description 116
    READ statement 383
  random
    DELETE statement 304
    description 116
    READ statement 383
  sequential
    DELETE statement 303
    description 116
    READ statement 380
ACCESS MODE clause 115
ACOS function 458
ADD statement
  common phrases 264
  CORRESPONDING phrase 284
  description and format 282
ADDRESS OF special register 10
ADDRESS OF under AIX and OS/2 403
ADVANCING phrase 437
AFTER phrase
  INSPECT statement 334
  PERFORM statement 368
  with REPLACING 331
  with TALLYING 329
  WRITE statement 438
AIX COBOL language differences
  ACCEPT statement 277
  APPLY WRITE-ONLY 129
  ASSIGN clause 107
  basis-names, library-names, text-names 40
  BLOCK CONTAINS 149
  CLOSE statement 296
  CODE-SET 159
  COMP-5 data item 211

© Copyright IBM Corp. 1991, 1998

AIX COBOL language differences *(continued)*
compiler limits 544
control characters 142
DBCS 2
environment-name 279
file handling 129
file status data-name-8 122
LABEL RECORDS 154
LINE SEQUENTIAL file I-O 113
locale definition 568
LOCK MODE clause 120
MORE-LABELS, GO TO 322
pointer data item size 214
RECORD CONTAINS clause 151
RECORDING MODE clause 157
RERUN clause 125
RESERVE clause 110
SAME AREA clause 127
SAME SORT AREA clause 128
SAME SORT-MERGE AREA clause 129
SET statement 403
SORT-CONTROL special register 16
SORT-FILE-SIZE under AIX and OS/2 17
SORT-MESSAGE under AIX and OS/2 18
SORT-MODE-SIZE under AIX and OS/2 18
status key value and meaning 271
USE...AFTER...LABEL PROCEDURE 535
WRITE ADVANCING 91, 437
aligning data 202
ALL
phrase of INSPECT statement 329, 331
SEARCH statement 396
UNSTRING statement 429
ALL literal
STOP statement 418
STRING statement 420
UNSTRING statement 429
ALL subscripting 451
ALPHABET clause 92
alphabet-name
description 92
MERGE statement 348
PROGRAM COLLATING SEQUENCE clause 88
SORT statement 410
alphabetic character in ACCEPT 277
alphabetic class and category 140
ALPHABETIC class test 240
alphabetic item
alignment rules 141
elementary move rules 354
alphabetic item *(continued)*
PICTURE clause 186
ALPHABETIC-LOWER class test 240
ALPHABETIC-UPPER class test 241
alphanumeric arguments 449
alphanumeric class and category
alignment rules 141
description 140
alphanumeric functions 448
alphanumeric item
alignment rules 141
elementary move rules 354
PICTURE clause 187
alphanumeric literal, control character restrictions 142
alphanumeric-edited item
alignment rules 141
elementary move rules 354
PICTURE clause 188
ALSO phrase
ALPHABET clause 94
EVALUATE statement 313
ALTER statement
description and format 285
GO TO statement and 322
segmentation considerations 286
altered GO TO statement 322
ALTERNATE RECORD KEY clause 118
AND logical operator 255
ANNUITY function 459
APPLY WRITE-ONLY clause 129
Area A (cols. 8-11) 33
Area B (cols. 12-72) 34
arguments 449
arithmetic expression
COMPUTE statement 300
description 233
EVALUATE statement 314
relation condition 243
arithmetic operator
description 234
permissible symbol pairs 235
arithmetic statements
ADD 282
common phrases 264
COMPUTE 300
DIVIDE 309
list of 268
multiple results 269
MULTIPLY 357
operands 268

arithmetic statements *(continued)*
programming notes 269
SUBTRACT 425
ASCENDING KEY phrase
collating sequence 174
description 345
MERGE statement 345
OCCURS clause 173
SORT statement 407
ASCII
collating sequence 551
processing considerations 565
specifying in SPECIAL-NAMES paragraph 92
ASIN function 460
ASSIGN clause
description 106
format 102
SELECT clause and 106
assigning index values 400
assignment-name
ASSIGN clause 106
RERUN clause 125
under AIX and OS/2 107
asterisk (*)
comment line 38
insertion character 193
at end condition
READ statement 382
RETURN statement 388
AT END phrase
READ statement 378
RETURN statement 388
SEARCH statement 394
AT END-OF-PAGE phrases 439
ATAN function 461
AUTHOR paragraph
description 83
format 74

B

B
insertion character 190
symbol in PICTURE clause 179
BASIS statement 512
basis-names on AIX and OS/2 40
batch compile 65
BEFORE phrase
INSPECT statement 334
PERFORM statement 368
BEFORE phrase *(continued)*
with REPLACING 331
with TALLYING 329
WRITE statement 438
binary arithmetic operators 234
binary data item, DISPLAY statement 305
BINARY phrase in USAGE clause 210
binary search 396
blank line 38
BLANK WHEN ZERO clause
description and format 164
USAGE IS INDEX clause 213
BLOCK CONTAINS clause
description 149
format 144
branching
GO TO statement 321
out-of-line PERFORM statement 366
BY CONTENT phrase
CALL statement 289
on INVOKE statement 339
BY REFERENCE phrase
CALL statement 289
on INVOKE statement 338
BY VALUE phrase
CALL statement 290
on INVOKE statement 339

C

C01-C012 under AIX and OS/2 437
call convention 539
CALL statement
CANCEL statement and 294
description and format 287
Linkage Section 230
ON OVERFLOW phrase 287
Procedure Division header 227, 230
program termination statements 287
subprogram linkage 287
transfer of control 57
USING phrase 230
called and calling programs, description 287
CALLINTERFACE directive 539
CANCEL statement 294
carriage control character 438
category of data
alphabetic items 186
alphanumeric items 187
alphanumeric-edited items 188

category of data *(continued)*
DBCS items 188
numeric items 186
numeric-edited items 187
relationship to class of data 140
CBL (PROCESS) statement 513
century window
See also date field
definition 61
CHAR function 462
character code set, specifying 92
character-string
COBOL word 3
representation in PICTURE clause 185
size determination 142
CHARACTERS BY phrase 331
CHARACTERS phrase
BLOCK CONTAINS clause 150
INSPECT statement 329
MEMORY SIZE clause 88
USAGE clause and 150
characters, valid in COBOL program 2
checkpoint processing, RERUN clause 125
class 69
CLASS clause 95
class condition 239, 241
class Data Division 132
class definition
affect of SELF and SUPER 337
class procedure division 225
CLASS-ID paragraph 79
Configuration Section 86
Data Division 132
description 69
Identification Division 76
inheritance rules 80
requirements for indexed tables 175
class identification division 79
class name, OO 41
class procedure division 225
CLASS-ID paragraph 79
class-name class test 241
classes of data 140
clauses 30, 31
CLOSE statement
affect on record locking 121
format and description 296
COBOL
class definition 69
language structure 2
COBOL *(continued)*
method definition 71
program structure 64
reference format 32
COBOL 74 Standard 573
COBOL 85 Standard
definition 573
COBOL word 3
CODE-SET clause
ALPHABET clause and 94
description 159
format 144
NATIVE phrase and 159
collating sequence
ASCENDING/DESCENDING KEY phrase and 174
ASCII 551
EBCDIC 548
locale definition 568
specified in OBJECT-COMPUTER paragraph 88
specified in SPECIAL-NAMES paragraph 92
COLLATING SEQUENCE phrase 88
ALPHABET clause 92
MERGE statement 347
SORT statement 410
colon character
description 29
required use of 522
column 7
indicator area 35
specifying comments 37
combined condition
description 256
evaluation rules 257
logical operators and evaluation results 257
order of evaluation 258
permissible element sequences 256
comma (,)
Configuration Section 86
DECIMAL-POINT IS COMMA clause 97
insertion character 190
symbol in PICTURE clause 181
comment line
description 37
Identification Division 83
in library text 518
COMMON clause 78
common processing facilities 270
COMP-1 through COMP-5 data items 211
comparison
cycle, INSPECT statement 335

comparison *(continued)*
DBCS operands 253
in EVALUATE statement 315
nonnumeric operands 249
numeric and nonnumeric operands 252
numeric operands 248
of index data items 252
of index-names 252
rules for COPY statement 520
compatible date field
See also date field
definition 60
compile-time switch 556
compiler directing statements
BASIS 512
COPY 516
DELETE 523
EJECT 524
ENTER 524
INSERT 525
READY TRACE 526
REPLACE 527
RESET TRACE 526
SERVICE LABEL 530
SERVICE RELOAD 531
SKIP1/2/3 531
TITLE 532
USE 533
compiler limits 544
compiler options
ADV 439
controlling output from 514
DATEPROC 59
NUMPROC 254
specifying 513
THREAD 175
TRUNC 143
complex conditions
abbreviated combined relation 258
combined condition 256
description 255
negated simple 256
complex OCCURS DEPENDING ON (CODO) 178
composite of operands 268
COMPUTATIONAL data items 210
COMPUTE statement
common phrases 266
description and format 300
computer-name 87, 88
condition
abbreviated combined relation 258
class 239
combined 256
complex 255
condition-name 242
EVALUATE statement 314
IF statement 323
negated simple 256
PERFORM UNTIL statement 368
relation 243
SEARCH statement 395
sign 253
simple 239
switch-status 254
condition-name
and conditional variable 162
description and format 242
rules for values 220
SEARCH statement 398
SET statement 402
SPECIAL-NAMES paragraph 92
switch status condition 92
conditional expression
comparing index-names and index data items 252
comparison of DBCS operands 253
description 239
order of evaluation of operands 257
parentheses in abbreviated combined relation conditions 259
conditional statements
description 262
GO TO statement 321
IF statement 323
list of 262
PERFORM statement 368
conditional variable 162
Configuration Section
description (programs, classes, methods) 86
REPOSITORY paragraph 98
SOURCE-COMPUTER paragraph 87
SPECIAL-NAMES paragraph 89
conformance rules
general rules 82
INVOKE...USING 341
multiple inheritance 80
SET...USAGE OBJECT REFERENCE 405
Contained Programs 64
continuation
area 32

continuation *(continued)*
lines 35, 37
CONTINUE statement 302
control characters, using on AIX or OS/2 142
CONTROL statement (*CONTROL) 514
control transfer 56
conversion of data, DISPLAY statement 305
CONVERTING phrase 333
COPY statement
comparison rules 520
description and format 516
example 521
replacement rules 520
REPLACING phrase 518
SUPPRESS option 518
CORRESPONDING (CORR) phrase
ADD statement 284
description 284
MOVE statement 352
SUBTRACT statement 426
with ON SIZE ERROR phrase 268
COS function 463
COUNT IN phrase, UNSTRING statement 430
CR (credit)
insertion character 191
symbol in PICTURE clause 182
cs (currency symbol)
in PICTURE clause 179
CURRENCY SIGN clause
description 96
Euro currency sign 96
restrictions on using NUMVAL-C function 488
currency sign value 96
currency symbol
in PICTURE clause 182
specifying in CURRENCY SIGN clause 96
currency symbol, default ($) 191
CURRENT-DATE function 464

D

data
alignment 141
categories 141, 185
classes 140
format of standard 142
hierarchies used in qualification 137
organization 111
signed 143
truncation of 142, 171
data category
alphabetic items 186
alphanumeric items 187
alphanumeric-edited items 188
DBCS items 188
numeric items 186
numeric-edited items 187
data conversion, DISPLAY statement 305
data description entry
BLANK WHEN ZERO clause 164
data-name 163
DATE FORMAT clause 164
description and format 161
FILLER phrase 163
GLOBAL clause 170
indentation and 140
JUSTIFIED clause 171
level-66 format (previously defined items) 162
level-88 format (condition-names) 162
level-number description 162
OCCURS clause 172
OCCURS DEPENDING ON (ODO) clause 176
PICTURE clause 178
REDEFINES clause 195
RENAMES clause 198
SIGN clause 200
SYNCHRONIZED clause 202
USAGE clause 209
VALUE clause 217
DATA DIVISION
ASCII considerations 567
data description entry 161
data relationships 137
data types 136
description (programs, classes, methods) 132
file description (FD) entry 148
levels of data 137
Linkage Section 135
sort description (SD) entry 148
Working-Storage Section 133
data flow
STRING statement 422
UNSTRING statement 432
data item
data description entry 161
description entry definition 133
EXTERNAL clause 170
record description entry 161
data manipulation statements
ACCEPT 277

data manipulation statements *(continued)*
INITIALIZE 325
list of 269
MOVE 352
overlapping operands 270
READ 376
RELEASE 385
RETURN 387
REWRITE 389
SET 400
STRING 420
UNSTRING 428
WRITE 436
data organization
access modes and 116
indexed 112
line sequential 112
relative 112
sequential 111
DATA RECORDS clause
description 155
format 144
data transfer 277
data types
file data 136
program data 137
data-name
data description entry 163
data-names
precedence if duplicate 132
DATE 280
date field
addition 236
arithmetic 235
compatible 60
DATE FORMAT clause 164
DATEPROC compiler option 59
DATEVAL function 468
definition 59
expansion of windowed date fields before use 165
group items that are date fields 167
in relation conditions 244
in sign conditions 254
MOVE statement, behavior in 356
non-date 61
purpose 58
restrictions 166
size errors 237, 266
storing arithmetic results 237
subtraction 237
date field *(continued)*
trigger values 166
UNDATE function 503
windowed date field conditional variables 243
date format
See also DATE FORMAT clause
definition 60
DATE FORMAT clause 164
combining with other clauses 167
DATE YYYYMMDD 280
DATE-COMPILED paragraph
description 83
format 74
DATE-OF-INTEGER function 466
DATE-TO-YYYYMMDD function 467
DATE-WRITTEN paragraph
description 83
format 74
DATEPROC compiler option 59
DATEVAL function 468
DAY 281
DAY YYYYDDD 281
DAY-OF-INTEGER function 470
DAY-OF-WEEK 281
DAY-TO-YYYYDDD function 471
DB (debit)
insertion character 191
symbol in PICTURE clause 182
DBCS (Double-Byte Character Set)
See also multi-byte characters
class and category 140
elementary move rules 355
PICTURE clause and 188
use with relational operators 246
using in comments 84
DBCS class condition 241
DD statements
See environment variables
De-editing 354
DEBUG-ITEM special register 11
debugging 555
DEBUGGING declarative 537
debugging line 38, 87
DEBUGGING MODE clause 87
decimal point (.) 266
DECIMAL-POINT IS COMMA clause
description 97
declarative procedures
description and format 230
PERFORM statement 365

declarative procedures *(continued)*
USE statement 230
declaratives
EXCEPTION/ERROR 533
LABEL 535
precedence rules for nested programs 535
USE FOR DEBUGGING 537
DECLARATIVES key word
begin in Area A 34
description 230
Declaratives Section 230
DELETE statement
affect on record locking 121
description and format 523
dynamic access 304
format and description 303
INVALID KEY phrases 304
random access 304
sequential access 303
DELIMITED BY phrase
STRING 421
UNSTRING statement 428
delimited scope statement 263
delimiter
INSPECT statement 332
UNSTRING statement 428
DELIMITER IN phrase, UNSTRING statement 430
DEPENDING phrase
GO TO statement 321
OCCURS clause 176
DESCENDING KEY phrase 173
collating sequence 174
description 345
MERGE statement 345
SORT statement 407
DISPLAY phrase in USAGE clause 211
DISPLAY statement
description and format 305
external 142, 189
programming notes 307
DIVIDE statement
common phrases 266
description and format 309
REMAINDER phrase 311
division header
format, Environment Division 86
format, Identification Division 74
format, Procedure Division 227
specification of 33
DO-UNTIL structure, PERFORM statement 368
DO-WHILE structure, PERFORM statement 368
Double-Byte Character Set (DBCS)
See also multi-byte characters
class and category 140
PICTURE clause and 188
use with relational operators 246
using in comments 84
DOWN BY phrase, SET statement 401
duplicate data-names, precedence 132
DUPLICATES phrase
KEY phrase 415
SORT statement 409
START statement 415
dynamic access mode
data organization and 116
DELETE statement 304
description 116
READ statement 383

E

E, symbol in PICTURE clause 180
EBCDIC
CODE-SET clause and 159
collating sequence 548
specifying in SPECIAL-NAMES paragraph 92
editing
fixed insertion 191
floating insertion 192
replacement 193
signs 143
simple insertion 190
special insertion 191
suppression 193
editing sign control symbol 182
eject page 38
EJECT statement 524
elementary item
alignment rules 141
basic subdivisions of a record 137
classes and categories 140
MOVE statement 353
nonnumeric operand comparison 252
size determination in program 142
size determination in storage 142
elementary move rules 353
ELSE NEXT SENTENCE phrase 323
END DECLARATIVES key word 230

END PROGRAM 65
end program header 34
END-CALL phrase 293
END-IF phrase 323
END-INVOKE phrase 343
end-of-file processing 296
END-OF-PAGE phrases 439
END-PERFORM phrase 367
ENTER statement 524
entry
  definition 30
ENTRY statement
  description and format 312
  subprogram linkage 312
Environment Division
  ASCII considerations 565
  compiler limits 544
  Configuration Section
    ALPHABET clause 92
    CURRENCY SIGN clause 96
    OBJECT-COMPUTER paragraph 88
    REPOSITORY paragraph 98
    SOURCE-COMPUTER paragraph 87
    SPECIAL-NAMES paragraph 89, 95
    SYMBOLIC CHARACTERS clause 95
  Input-Output Section
    FILE-CONTROL paragraph 102
  REPOSITORY paragraph 98
environment names with WRITE ADVANCING 437
environment variables
  in ACCEPT statement 279
  in DISPLAY statement 305
environment-name
  SPECIAL-NAMES paragraph 91
environment-name under AIX and OS/2 279
EOP phrases 439
equal sign (=) 243
EQUAL TO relational operator 243
ERROR declarative statement 533
EUC
  description 2
Euro currency sign
  specifying in CURRENCY SIGN clause 96
EVALUATE statement
  comparing operands 315
  determining truth value 314
  format and description 313
evaluation rules
  combined conditions 257
  EVALUATE statement 315
evaluation rules *(continued)*
  nested IF statement 324
EXCEPTION declarative statement 533
EXCEPTION/ERROR declarative
  CLOSE statement 297
  DELETE statement 303
  description and format 533
execution flow
  ALTER statement changes 285
  PERFORM statement changes 365
EXIT METHOD statement
  format and description 318
EXIT PROGRAM statement
  format and description 319
EXIT statement
  format and description 317
  PERFORM statement 366
expanded date field
  See also date field
  definition 60
expanded year
  See also date field
  definition 60
expansion of windowed date fields before use 165
explicit
  scope terminators 264
exponentiation
  exponential expression 234
expression, arithmetic 233
EXTEND phrase
  OPEN statement 359
EXTERNAL clause
  with data item 170
  with file name 148
external decimal item
  DISPLAY statement 305
external floating point
  alignment rules 142
  DISPLAY statement 305
  PICTURE clause and 189

F

FACTORIAL function 472
FALSE phrase 314
FD (File Description) entry
  BLOCK CONTAINS clause 149
  DATA RECORDS clause 155
  description 147
  format 144

FD (File Description) entry *(continued)*
LABEL RECORDS clause 154
level indicator 137
VALUE OF clause 155
figurative constant
DISPLAY statement 306
STOP statement 418
STRING statement 420
symbolic-character 9
UNSTRING statement 429
file
data type 136
definition 136
handling, on AIX and OS/2 129
labels 154
file description entry
GLOBAL clause 149
file organization
definition 116
LINAGE clause 155
line sequential 112
types of 111
file position indicator
description 276
READ statement 382
File Section
EXTERNAL clause 148
for object-oriented programs 132
RECORD clause 151
file sharing 120
FILE STATUS clause
DELETE statement and 303
description 122
format 102
INVALID KEY phrase and 274
status key 270
FILE-CONTROL paragraph
ASSIGN clause 106
description and format 102
FILE STATUS clause 122
ORGANIZATION clause 111
PADDING CHARACTER clause 114
RECORD KEY clause 117
RELATIVE KEY clause 119
RESERVE clause 110
SELECT clause 106
file-name, specifying on SELECT clause 106
FILLER phrase
CORRESPONDING phrase 163
data description entry 163
fixed insertion editing 191
fixed-length
item, maximum length 161
records 149
floating insertion editing 192
floating-point
DISPLAY statement 305
internal 142
FOOTING phrase of LINAGE clause 155
FOR REMOVAL phrase 296, 297
format notation, rules for xiii
FROM phrase
ACCEPT statement 278
REWRITE statement 389
SUBTRACT statement 425
with identifier 275
WRITE statement 437
function
arguments 449
class and category 141
description 447
rules for usage 449
types of functions 448

G

G, symbol in PICTURE clause 180
GIVING phrase
ADD statement 282
arithmetic 266
DIVIDE statement 311
MERGE statement 349
MULTIPLY statement 357
SORT statement 411
SUBTRACT statement 426
GLOBAL clause
with data item 170
with file name 149
GO TO statement
altered 322
conditional 321
format and description 321
MORE-LABELS 322
SEARCH statement 394
unconditional 321
GOBACK statement 320
GREATER THAN OR EQUAL TO symbol (>=) 243
GREATER THAN symbol (>) 243
group item
class and categories 140

group item *(continued)*
description 137
MOVE statement 356
nonnumeric operand comparison 252
group move rules 356

H

halting execution 418
HIGH-VALUE(S) figurative constant 94
hyphen (-), in indicator area 35

I

IBM extensions, format description xiii
Identification Division
CLASS-ID paragraph 79
format (program, class, method) 74
METHOD-ID paragraph 81
optional paragraphs 83
PROGRAM-ID paragraph 77
identifier 46, 233
IF statement 323
imperative statement 261
implicit
redefinition of storage area 148, 195
scope terminators 264
in line PERFORM statement 365
indentation 35, 140
index
data item 252, 352
relative indexing 52
SET statement 52
index name
assigning values 400
comparisons 252
data item definition 212
OCCURS clause 175
PERFORM statement 375
SEARCH statement 393
SET statement 400
INDEX phrase in USAGE clause 212
INDEXED BY phrase 175
indexed files
CLOSE statement 297
DELETE statement 304
FILE-CONTROL paragraph format 102
I-O-CONTROL paragraph format 124
organization 112
permissible statements for 364
indexed files *(continued)*
READ statement 382
START statement 416
indexed organization
description 112
FILE-CONTROL paragraph format 102
I-O-CONTROL paragraph format 124
indexing
description 51
MOVE statement evaluation 352
OCCURS clause 51, 172
relative 52
SET statement and 52
indicator area 32
industry specifications 571
INHERITS clause 79
INITIAL clause 78
initial state of program 78
INITIALIZE statement
format and description 325
overlapping operands, unpredictable results 269
input file, label processing 362
Input-Output Section
description 100
FILE-CONTROL paragraph 102
format 100
I-O-CONTROL paragraph 124
input-output statements
ACCEPT 277
CLOSE 296
common processing facilities 270
DELETE 303
DISPLAY 305
EXCEPTION/ERROR procedures 534
general description 270
OPEN 359
READ 376
REWRITE 389
START 415
WRITE 436
INPUT phrase
OPEN statement 359
USE statement 533
INPUT PROCEDURE phrase
RELEASE statement 385
SORT statement 411
Input-Output Section
on AIX and OS/2 113
insertion editing
fixed (numeric-edited items) 191

insertion editing *(continued)*
floating (numeric-edited items) 192
simple 190
special (numeric-edited items) 191
INSPECT statement
AFTER phrase 332
BEFORE phrase 332
comparison cycle 335
CONVERTING phrase 333
overlapping operands, unpredictable results 269
REPLACING phrase 329
INSTALLATION paragraph
description 83
format 74
instance data 71
integer arguments 449
INTEGER function 473
Integer functions 448
INTEGER-OF-DATE function 474
INTEGER-OF-DAY function 475
INTEGER-PART function 476
internal floating-point
alignment rules 142
DISPLAY statement 305
INTO phrase
DIVIDE statement 309
READ statement 376
RETURN statement 387
STRING statement 421
UNSTRING statement 430
with identifier 275
intrinsic functions
ACOS 458
alphanumeric function 448
ANNUITY 459
ASIN 460
ATAN 461
CHAR 462
COS 463
CURRENT-DATE 464
DATE-OF-INTEGER 466
DATE-TO-YYYYMMDD 467
DATEVAL 468
DAY-OF-INTEGER 470
DAY-TO-YYYYDDD 471
FACTORIAL 472
floating-point literals 451
INTEGER 473
integer function 448
INTEGER-OF-DATE 474
intrinsic functions *(continued)*
INTEGER-OF-DAY 475
INTEGER-PART 476
LENGTH 477
LOG 478
LOG10 479
LOWER-CASE 480
MAX 481
MEAN 482
MEDIAN 483
MIDRANGE 484
MIN 485
MOD 486
numeric function 448
NUMVAL 487
NUMVAL-C 488
ORD 490
ORD-MAX 491
ORD-MIN 492
PRESENT-VALUE 493
RANDOM 494
RANGE 495
REM 496
REVERSE 497
SIN 498
SQRT 499
STANDARD-DEVIATION 500
SUM 501
summary of 455
TAN 502
UNDATE 503
UPPER-CASE 504
VARIANCE 505
WHEN-COMPILED 506
YEAR-TO-YYYY 508
YEARWINDOW 509
invalid key condition 274
INVALID KEY phrase
DELETE statement 304
READ statement 379
REWRITE statement 390
START statement 416
WRITE statement 440
INVOKE statement
BY CONTENT phrase 339
BY REFERENCE phrase 338
BY VALUE phrase 339
format and description 337
NOT ON EXCEPTION phrase 343
ON EXCEPTION phrase 342

INVOKE statement *(continued)*
RETURNING phrase 341
USING phrase 338
I-O-CONTROL paragraph
APPLY WRITE-ONLY clause 129
checkpoint processing in 125
description 100, 124
MULTIPLE FILE TAPE clause 129
order of entries 124
RERUN clause 125
SAME AREA clause 127
SAME RECORD AREA clause 127
SAME SORT AREA clause 128
SAME SORT-MERGE AREA clause 129
ISCII processing considerations 565

J

JUSTIFIED clause
description and format 171
effect on initial settings 172
STRING statement 421
truncation of data 171
USAGE IS INDEX clause and 171
VALUE clause and 218

K

Kanji 241
key of reference 112
KEY phrase
OCCURS clause 173
READ statement 378
SEARCH statement 393
SORT statement 407
START statement 415

L

LABEL declarative 535
label processing, OPEN statement 362
LABEL RECORDS clause
description 154
format 144
Language Environment Callable Services
description 287
LEADING phrase
INSPECT statement 329, 331
SIGN clause 201
LENGTH function 477
LENGTH OF special register 12
LESS THAN OR EQUAL TO symbol (<=) 243
LESS THAN symbol (<) 243
level
01 item 138
02-49 item 138
66 item 140
77 item 140
88 item 140
indicator, definition of 137
level number
definition 137
description and format 162
FILLER phrase 163
library-name
COPY statement 516
library-names
on AIX and OS/2 40
limit values, date field 166
limits of the compiler 544
LINAGE clause
description 155
diagram of phrases 156
format 144
LINAGE-COUNTER special register
description 13
WRITE statement 439
line advancing 437
line sequential file organization 112
LINE/LINES, WRITE statement 437
LINES AT BOTTOM phrase 156
LINES AT TOP phrase 156
Linkage Section
called subprogram 230
description 135
levels under AIX and OS/2 403
requirement for indexed items 175
VALUE clause 217
literal
and arithmetic expressions 233
ASSIGN clause 106
CODE-SET clause and ALPHABET clause 94
CURRENCY SIGN clause 96
description 20
nonnumeric operand comparison 252
null-terminated nonnumeric 23
STOP statement 418
VALUE clause 218

local storage
defining with RECURSIVE clause 77
requirement for indexed items 175
Local-Storage
object-oriented COBOL 132
locale 568
LOCK MODE clause 120
LOG function 478
LOG10 function 479
logical operator
complex condition 255
in evaluation of combined conditions 257
list of 255
logical record
definition 136
file data 136
program data 137
record description entry and 136
RECORDS phrase 150
LOW-VALUE(S) figurative constant 94
LOWER-CASE function 480

M

MAX function 481
maximum index value 52
MEAN function 482
MEDIAN function 483
MEMORY SIZE clause 88
MERGE statement
ASCENDING/DESCENDING KEY phrase 345
COLLATING SEQUENCE phrase 347
format and description 345
GIVING phrase 349
OUTPUT PROCEDURE phrase 350
USING phrase 349
METACLASS clause 79, 213
metaclass, description 69
method data division 132
method definition
affect of SELF and SUPER 337
Data Division 132
description 71
Identification Division 76
inheritance rules 80
method procedure division 225
METHOD-ID paragraph 81
method identification division 81
method name 40
method procedure division 225
METHOD-ID paragraph 81
methods
available to subclasses 80
exiting 318
invoking 337
recursively reentering 77
reusing 79
MIDRANGE function 484
millennium language extensions
syntax 58
millennium language extensions (MLE)
See also date field
description 58
MIN function 485
minus sign (-)
COBOL character 2
fixed insertion symbol 191
floating insertion symbol 192, 193
SIGN clause 201
mnemonic-name
ACCEPT statement 277
DISPLAY statement 306
SET statement 402
SPECIAL-NAMES paragraph 91
WRITE statement 438
MOD function 486
MORE-LABELS GO TO statement 322
MOVE statement
CORRESPONDING phrase 352
elementary moves 353
format and description 352
group moves 356
multi-byte characters
in COBOL words 4
in literals 20
MULTIPLE FILE TAPE clause 129
multiple inheritance 80
multiple record processing, READ statement 379
multiple results, arithmetic statements 269
multiple volume files, treatment on AIX and OS/2 129
MULTIPLY statement
common phrases 266
format and description 357
multivolume files
READ statement 382
WRITE statement 442

N

native binary data item 211
native character set 92
native collating sequence 92
negated combined condition 256
negated simple condition 256
NEGATIVE 254
nested IF structure
  description 324
  EVALUATE statement 313
nested programs
  description 64
  precedence rules for 535
NEXT RECORD phrase, READ statement 376
NEXT SENTENCE phrase
  IF statement 323
  SEARCH statement 394
NO ADVANCING phrase, DISPLAY statement 306
NO REWIND phrase
  OPEN statement 359
  under AIX and OS/2 296
non-date
  See also date field
  definition 61
non-reel file, definition 297
nonnumeric literals 23
nonnumeric operands, comparing 249
NOT AT END phrase
  READ statement 378
  RETURN statement 388
NOT INVALID KEY phrase
  DELETE statement 304
  READ statement 379
  REWRITE statement 390
  START statement 416
NOT ON EXCEPTION phrase
  CALL statement 292
  on INVOKE statement 343
NOT ON OVERFLOW phrase
  STRING statement 422
  UNSTRING statement 431
NOT ON SIZE ERROR phrase
  ADD statement 284
  DIVIDE statement 311
  general description 266
  MULTIPLY statement 358
  SUBTRACT statement 427
NULL 222
null block branch, CONTINUE statement 302
null-terminated nonnumeric literals 23
numeric arguments 449
numeric class and category 140
NUMERIC class test 240
numeric function 448
numeric item 186
numeric operands, comparing 248
numeric-edited item
  alignment rules 141
  editing signs 143
  elementary move rules 354
  PICTURE clause 187
NUMVAL function 487
NUMVAL-C function 488

O

Object Program 64
OBJECT REFERENCE phrase 213
object time switch 556
OBJECT-COMPUTER paragraph 88
object-oriented COBOL
  class definition 69
  comparison rules 248
  conformance rules
    general rules 82
    INVOKE...USING 341
    multiple inheritance 80
    SET...USAGE OBJECT REFERENCE 405
  Data Division (class and method) 132
  effect of GLOBAL attribute 133
  Identification Division (class and method) 74
  INHERITS clause 79
  INVOKE statement 337
  method definition 71
  method name 40
  multiple inheritance 80
  OO class name 41
  Procedure Division (class and method) 225
  REPOSITORY paragraph 98
  SELF and SUPER special characters 7
  specifying configuration section 86
  subclasses and methods 80
  USAGE OBJECT REFERENCE clause 213
objects in EVALUATE statement 313
obsolete language elements xii
OCCURS clause
  ASCENDING/DESCENDING KEY phrase 173
  description 172
  INDEXED BY phrase 175

OCCURS clause *(continued)*
restrictions 172
variable-length tables format 176
OCCURS DEPENDING ON (ODO) clause
complex 178
description 176
format 176
RECORD clause 151
REDEFINES clause and 172
SEARCH statement and 172
subject and object of 176
subject of 172
subscripting 49
OFF phrase, SET statement 402
OMITTED 289, 290
ON EXCEPTION phrase
CALL statement 292
on INVOKE statement 342
ON OVERFLOW phrase
CALL statement 293
DISPLAY statement 307
STRING statement 421, 432
ON phrase, SET statement 402
ON SIZE ERROR phrase
ADD statement 284
arithmetic statements 266
COMPUTE statement 301
DIVIDE statement 311
MULTIPLY statement 358
SUBTRACT statement 427
OPEN statement
affect on record locking 121
for new/existing files 360
format and description 359
I-O phrase 359
label processing 362
phrases 359
programming notes 362
system dependencies 364
operands
comparison of nonnumeric 249
comparison of numeric 248
composite of 268
overlapping 269, 270
operational sign
algebraic, description of 143
SIGN clause and 143
USAGE clause and 143
optional file
See SELECT OPTIONAL clause
ORD function 490
ORD-MAX function 491
ORD-MIN function 492
order of entries
clauses in FILE-CONTROL paragraph 102
IO CONTROL paragraph 124
order of evaluation in combined conditions 258
ORGANIZATION clause
description 111
format 102
ORGANIZATION IS INDEXED clause 111
ORGANIZATION IS LINE SEQUENTIAL clause 111
ORGANIZATION IS RELATIVE clause 111
ORGANIZATION IS SEQUENTIAL clause 111
OS/2 COBOL language differences
ACCEPT statement 277
APPLY WRITE-ONLY 129
ASSIGN clause 107
basis-names, library-names, text-names 40
BLOCK CONTAINS 149
CLOSE statement 296
CODE-SET clause 159
COMP-5 data item 211
compiler limits 544
control characters 142
DBCS 2
environment-name 279
file handling 129
file status data-name-8 122
LABEL RECORDS 154
LINE SEQUENTIAL file I-O 113
locale definition 568
LOCK MODE clause 120
MORE-LABELS, GO TO 322
pointer data item size 214
RECORD CONTAINS clause 151
RECORDING MODE clause 157
RERUN clause 125
RESERVE clause 110
SAME AREA clause 127
SAME SORT AREA clause 128
SAME SORT-MERGE AREA clause 129
SET statement 403
SORT-CONTROL special register 16
SORT-FILE-SIZE under AIX and OS/2 17
SORT-MESSAGE under AIX and OS/2 18
SORT-MODE-SIZE under AIX and OS/2 18
specifying call convention 539
status key value and meaning 271
USAGE clause 213

OS/2 COBOL language differences *(continued)*
USE...AFTER...LABEL PROCEDURE 535
WRITE ADVANCING 91, 437
out-of-line PERFORM statement 366
outermost programs, debugging 537
output file, label processing 362
OUTPUT phrase 359
OUTPUT PROCEDURE phrase
MERGE statement 350
RETURN statement 387
SORT statement 412
OVERFLOW phrase
CALL statement 293
STRING statement 421, 432
overlapping operands invalid in
arithmetic statements 269
data manipulation statements 270

P

P, symbol in PICTURE clause 180
PACKED-DECIMAL phrase in USAGE clause 210
PADDING CHARACTER clause 114
page eject 38
paragraph
description 30, 232
header, specification of 33
termination, EXIT statement 317
paragraph name
description 232
specification of 33
parentheses
combined conditions, use 257
in arithmetic expressions 234
partial listings 514
PASSWORD clause
description 120
system dependencies 120
PERFORM statement
branching 366
conditional 368
END-PERFORM phrase 367
EVALUATE statement 313
execution sequences 367
EXIT statement 317
format and description 365
in-line 366
out-of-line 366
TIMES phrase 367
VARYING phrase 369, 371
period (.)
actual decimal point 191
phrase, definition 31
physical record
BLOCK CONTAINS clause 149
definition 136
file data 136
file description entry and 136
RECORDS phrase 150
PICTURE clause
and class condition 240
computational items and 210
CURRENCY SIGN clause 96
data categories in 185
DECIMAL-POINT IS COMMA clause 97, 179
description 178
editing 190
format 178
sequence of symbols 183
symbols used in 179
PICTURE SYMBOL phrase 97
plus (+)
fixed insertion symbol 191
floating insertion symbol 192, 193
insertion character 193
SIGN clause 201
pointer data item
defined with USAGE clause 214
relation condition 247
SET statement 403
size on Workstation 214
POINTER phrase
STRING statement 421
UNSTRING statement 431
POSITIVE 254
PRESENT-VALUE function 493
PREVIOUS RECORD phrase, READ statement 376
print files, WRITE statement 442
procedure branching
GO TO statement 321
statements, executed sequentially 276
Procedure Branching Statements 276
Procedure Division
declarative procedures 230
format (programs, methods, classes) 225
header 227
statements 277
procedure-name
GO TO statement 321
MERGE statement 350

procedure-name *(continued)*
PERFORM statement 365
SORT statement 411
PROCEDURE-POINTER data item
defined with USAGE clause 215
relation condition 247
SET statement 404
size on Workstation 214
procedure, description 231
PROGRAM COLLATING SEQUENCE clause
ALPHABET clause 92
SPECIAL-NAMES paragraph and 88
program termination
GOBACK statement 320
STOP statement 418
PROGRAM-ID paragraph
description 77
format 74
program-name, rules for referencing 67
program, separately-compiled 64
programming notes
ACCEPT statement 277
altered GO TO statement 285
arithmetic statements 269
data manipulation statements 420, 428
DELETE statement 303
DISPLAY statement 307
EXCEPTION/ERROR procedures 535
OPEN statement 362
PERFORM statement 367
RECORDS clause 151
STRING statement 420
UNSTRING statement 428
programming structures 368
programs, recursive 77
pseudo-text
COPY statement operand 519
description 38
punch files, WRITE statement 442

Q

quotation mark (") character 35

R

railroad track format, how to read xiii
random access mode
data organization and 116
DELETE statement 304
random access mode *(continued)*
description 116
READ statement 383
RANDOM function 494
RANGE function 495
READ statement
affect on record locking 121
AT END phrases 378
dynamic access mode 383
format and description 376
INTO identifier phrase 275, 376
INVALID KEY phrases 274, 379
KEY phrase 378
multiple record processing 379
multivolume files 382
NEXT RECORD phrase 376
overlapping operands, unpredictable results 269
programming notes 384
random access mode 383
READY TRACE statement 526
receiving field
COMPUTE statement 300
MOVE statement 352
multiple results rules 269
SET statement 400
STRING statement 421
UNSTRING statement 430
record
area description 151
elementary items 137
fixed-length 149
logical, definition of 136
physical, definition of 136
RECORD clause
description and format 151
omission of 151
RECORD CONTAINS 0 CHARACTERS 151
record description entry
levels of data 138
logical record 136
RECORD KEY clause
description 117
format 102
record key in indexed file 304
record locking 120
RECORDING MODE clause 157
RECORDS phrase
BLOCK CONTAINS clause 150
RERUN clause 126

RECURSIVE clause 77
recursive methods 337
recursive programs 77
  requirement for indexed items 175
REDEFINES clause
  description 195
  examples of 198
  format 195
  general considerations 196
  OCCURS clause restriction 196
  SYNCHRONIZED clause and 202
  undefined results 198
  VALUE clause and 196
redefinition, implicit 148
REEL phrase 296, 297
reference-modification 52, 54
reference-modifier
  ALL subscripting 451
Reference, methods of
  Simple data 46
relation character
  COPY statement 519
  INITIALIZE statement 325
  INSPECT statement 329
relation condition
  abbreviated combined 258
  comparison of numeric and nonnumeric operands 248
  comparison with nonnumeric second operand 251
  comparison with numeric second operand 249
  description 243
  operands of equal size 250
  operands of unequal size 250
relational operator
  in abbreviated combined relation condition 258
  meaning of each 244
  relation condition use 243
relative files
  access modes allowed 117
  CLOSE statement 297
  DELETE statement 304
  FILE-CONTROL paragraph format 102
  I-O-CONTROL paragraph format 124
  organization 112
  permissible statements for 364
  READ statement 380
  RELATIVE KEY clause 117, 119
  REWRITE statement 391
  START statement 417
RELATIVE KEY clause
  description 119
  format 102
relative organization
  access modes allowed 117
  description 112
  FILE-CONTROL paragraph format 102
  I-O-CONTROL paragraph format 124
RELEASE statement 269, 385
REM function 496
REMAINDER phrase of DIVIDE statement 311
RENAMES clause
  description and format 198
  INITIALIZE statement 325
  level 66 item 140, 198
  PICTURE clause 178
REPLACE statement
  comparison operation 528
  continuation rules for pseudo-text 528
  description and format 527
  special notes 529
replacement editing 193
replacement rules for COPY statement 520
REPLACING phrase
  COPY statement 518
  INITIALIZE statement 326
REPOSITORY paragraph 98
required words xiii
RERUN clause
  checkpoint processing 125
  description 125
  format 124
  RECORDS phrase 125
  sort/merge 126
  under AIX and OS/2 125
RESERVE clause
  description 110
  format 102
  under AIX and OS/2 110
reserved word list 558
RESET TRACE statement 526
result field
  GIVING phrase 266
  NOT ON SIZE ERROR phrase 266
  ON SIZE ERROR phrase 266
  ROUNDED phrase 266
RETURN statement
  AT END phrase 388
  description and format 387
  overlapping operands, unpredictable results 269

RETURN-CODE special register 14
RETURNING phrase
  CALL statement 291
  on INVOKE statement 341
reusing logical records 390
REVERSE function 497
REWRITE statement
  affect on record locking 121
  description and format 389
  FROM identifier phrase 275
  INVALID KEY phrase 390
ROUNDED phrase
  ADD statement 284
  COMPUTE statement 301
  description 266
  DIVIDE statement 311
  MULTIPLY statement 358
  size error checking and 267
  SUBTRACT statement 427
rules for syntax notation xiii
Rules for Usage 449
run unit
  description 64
  termination with CANCEL statement 295

S

S 181
S01-S05 environment names under AIX and OS/2 437
SAME AREA clause under AIX and OS/2 127
SAME clause 127
SAME RECORD AREA clause
  description 127
  format 124
SAME SORT AREA clause
  description 128
  format 124
SAME SORT AREA clause under AIX and OS/2 128
SAME SORT-MERGE AREA clause
  description 129
  format 124
scope terminator
  explicit 264
  implicit 264
SD (Sort File Description) entry
  Data Division 148
  DATA RECORDS clause 155
  description 144, 147
  level indicator 137
SEARCH statement
  AT END phrase 394
  binary search 396
  description and format 393
  serial search 394
  SET statement 394
  USAGE IS INDEX clause 213
  VARYING phrase 395
  WHEN phrase 394
section 30, 231
section header
  description 231
  specification of 33
section name
  description 232
  in EXCEPTION/ERROR declarative 533
SECURITY paragraph
  description 83
  format 74
segmentation considerations 286
SELECT clause
  ASSIGN clause and 106
  format 102
  specifying a file name 106
SELECT OPTIONAL clause
  CLOSE statement 297
  description 106
  format 102
  specification for sequential I-O files 106
selection objects in EVALUATE statement 313
selection subjects in EVALUATE statement 313
SELF special character word 7, 337
sending field
  MOVE statement 352
  SET statement 400
  STRING statement 420
  UNSTRING statement 428
sentence
  COBOL, definition 31
  description 233
SEPARATE CHARACTER phrase of SIGN clause 201
separate sign, class condition 240
separately-compiled program 64
separator 221
sequence number area (cols. 1-6) 32
sequential access mode
  data organization and 116
  DELETE statement 303
  description 116
  READ statement 380

sequential access mode *(continued)*
REWRITE statement 390
sequential files
access mode allowed 116
CLOSE statement 296, 297
description 111
file description entry 144
FILE-CONTROL paragraph format 102
LINAGE clause 156
OPEN statement 359
PASSWORD clause valid with 120
permissible statements for 363
READ statement 380
REWRITE statement 390
SELECT OPTIONAL clause 106
serial search
PERFORM statement 369
SEARCH statement 394
SERVICE LABEL statement 530
SERVICE RELOAD statement 531
SET statement
description and format 400
DOWN BY phrase 401
index data item values assigned 212
OFF phrase 402
ON phrase 402
overlapping operands, unpredictable results 269
pointer data item 403
procedure-pointer data item 404
requirement for indexed items 175
SEARCH statement 401
TO phrase 400
TO TRUE phrase 402
UP BY phrase 401
USAGE IS INDEX clause 213
USAGE OBJECT REFERENCE 405
sharing data 171
sharing files 149
SHIFT-OUT, SHIFT-IN special registers 15
Sibling program 64
SIGN clause 200
sign condition 253
SIGN IS SEPARATE clause 201
signed
numeric item, definition 186
operational signs 143
simple condition
combined 256
description and types 239
negated 256
Simple data reference 46
simple insertion editing 190
SIN function 498
size-error condition 266
skip to next page 38
SKIP1/2/3 statement 531
slack bytes
between 207
within 204
slash (/)
comment line 37
insertion character 190
symbol in PICTURE clause 181
SOMClass, root for metaclasses 80
SOMObject, root for classes 80
Sort File Description entry
See SD (Sort File Description) entry
SORT statement
ASCENDING KEY phrase 407
COLLATING SEQUENCE phrase 410
DESCENDING KEY phrase 407
description and format 407
DUPLICATES phrase 409
GIVING phrase 411
INPUT PROCEDURE phrase 411
OUTPUT PROCEDURE phrase 412
USING phrase 411
SORT-CONTROL special register 16
SORT-CORE-SIZE special register 17
SORT-FILE-SIZE special register 17
SORT-MESSAGE special register 18
SORT-MODE-SIZE special register 18
SORT-RETURN special register 18
Sort/Merge feature
I-O-CONTROL paragraph format 124
MERGE statement 345
RELEASE statement 385
RERUN clause 126
RETURN statement 387
SAME SORT AREA clause 128
SAME SORT-MERGE AREA clause 129
SORT statement 407
Sort/Merge file statement phrases
ASCENDING/DESCENDING KEY phrase 345
COLLATING SEQUENCE phrase 347
GIVING phrase 349
OUTPUT PROCEDURE phrase 350
USING phrase 349
source code listing 515

source language debugging 555
source program
  library, programming notes 521
  standard COBOL reference format 32
SOURCE-COMPUTER paragraph 87
special insertion editing 191
special registers
  ADDRESS OF 10
  DEBUG-ITEM 11
  LENGTH OF 12
  LINAGE-COUNTER 13
  RETURN-CODE 14
  SHIFT-OUT, SHIFT-IN 15
  SORT-CONTROL 16
  SORT-CORE-SIZE 17
  SORT-FILE-SIZE 17
  SORT-MESSAGE 18
  SORT-MODE-SIZE 18
  SORT-RETURN 18
  TALLY 19
  WHEN-COMPILED 19
SPECIAL-NAMES paragraph
  ACCEPT statement 278
  ALPHABET clause 92
  ASCII-encoded file specification 159
  CLASS clause 95
  CODE-SET clause and 159
  CURRENCY SIGN clause 96
  DECIMAL-POINT IS COMMA clause 97
  description 89
  format 89
  mnemonic names 91
specifications 571
SQRT function 499
standard alignment
  JUSTIFIED clause 172
  rules 141
standard COBOL format 32
standard data format 142
STANDARD-1 phrase 92
STANDARD-2 phrase 92
STANDARD-DEVIATION function 500
standards 571
START statement
  affect on record locking 121
  description and format 415
  indexed file 416
  INVALID KEY phrase 274, 416
  relative file 417
  status key considerations 416
statement
  categories of 261
  conditional 262
  data manipulation 269
  delimited scope 263
  description 31, 233
  imperative 261
  input-output 270
  procedure branching 276
statement operations
  common phrases 264
  file position indicator 276
  INTO/FROM identifier phrase 275
status key
  common processing facility 270
  file processing 534
  value and meaning 271
STOP RUN statement 418
STOP statement 418
storage
  map listing 515
  MEMORY SIZE clause 88
  REDEFINES clause 195
STRING statement
  description and format 420
  execution of 422
  overlapping operands, unpredictable results 269
structure of the COBOL language 2
structured programming
  DO-WHILE and DO-UNTIL 368
subclasses and methods 80
subjects in EVALUATE statement 313
subprogram linkage
  CALL statement 287
  CANCEL statement 294
  ENTRY statement 312
subprogram termination
  CANCEL statement 294
  EXIT PROGRAM statement 319
  GOBACK statement 320
subscripting
  definition and format 49
  INDEXED BY phrase of OCCURS clause 175
  MOVE statement evaluation 352
  OCCURS clause specification 172
  table references 49
  using data-names 51
  using index-names (indexing) 51
  using integers 51

substitution field of INSPECT REPLACING 329
SUBTRACT statement
common phrases 264
description and format 425
SUM function 501
SUPER special character word 7, 337
SUPPRESS option, COPY 518
suppress output 514
suppression editing 193
switch-status condition 254
SYMBOLIC CHARACTERS clause 95
symbols in PICTURE clause 179
SYNCHRONIZED clause 202
VALUE clause and 218
syntax notation, rules for xiii
system considerations, subprogram linkage
CALL statement 287
CANCEL statement 294
system information transfer, ACCEPT statement 279
system input device, ACCEPT statement 278
system-name 88
computer-name 87
SOURCE-COMPUTER paragraph 87

T

tab character, restriction in IBM COBOL 28
table references
indexing 51
subscripting 49
TALLY special register 19
TALLYING phrase
INSPECT statement 329
UNSTRING statement 431
TAN function 502
termination of execution
EXIT METHOD statement 318
EXIT PROGRAM statement 319
GOBACK statement 320
STOP RUN statement 418
terminators, scope 264
text words 517
text-name
literal-1 516
under AIX and OS/2 40
THREAD compiler option 175
requirement for indexed items 175
THROUGH (THRU) phrase
ALPHABET clause 93
CLASS clause 96
THROUGH (THRU) phrase *(continued)*
EVALUATE statement 314
PERFORM statement 365
RENAMES clause 198
VALUE clause 219
TIME 281
TIMES phrase of PERFORM statement 367
TITLE statement 532
TO phrase, SET statement 400
TO TRUE phrase, SET statement 402
transfer of control
ALTER statement 286
explicit 56
GO TO statement 321
IF statement 324
implicit 56
PERFORM statement 365
transfer of data
ACCEPT statement 277
MOVE statement 352
STRING statement 420
UNSTRING statement 428
trigger values, date field 166
truncation of data
arithmetic item 142
JUSTIFIED clause 171
ROUNDED phrase 266
TRUNC compiler option 143
truth value
complex conditions 255
EVALUATE statement 314
IF statement 323
of complex condition 255
sign condition 254
with conditional statement 262
type conformance
general rules 82
INVOKE...USING 341
multiple inheritance 80
SET...USAGE OBJECT REFERENCE 405

U

unary operator 234
unconditional GO TO statement 321
UNDATE function 503
unit file, definition 297
UNIT phrase 296
universal object reference 213

unsigned numeric item, definition 186
UNSTRING statement
  description and format 428
  execution 432
  overlapping operands, unpredictable results 269
  receiving field 430
  sending field 428
UP BY phrase, SET statement 401
UPON phrase, DISPLAY 306
UPPER-CASE function 504
UPSI-0 through UPSI-7, program switches
  and switch-status condition 255
  condition-name 92
  processing special conditions 91
  SPECIAL-NAMES paragraph 91
USAGE clause
  BINARY phrase 210
  CODE-SET clause and 159
  COMPUTATIONAL phrases 211
  description 209
  DISPLAY phrase 211
  DISPLAY-1 phrase 212
  elementary item size 142
  format 209
  INDEX phrase 212
  operational signs and 143
  PACKED-DECIMAL phrase 210
  USAGE IS PROCEDURE-POINTER 215
  VALUE clause and 218
USAGE DISPLAY
  class condition identifier 239
  STRING statement and 420
USAGE IS COMPUTATIONAL phrases 211
USAGE IS OBJECT REFERENCE syntax 209
USAGE IS POINTER 214
USAGE IS PROCEDURE-POINTER 215
USAGE OBJECT REFERENCE phrase 337
USE...AFTER...LABEL PROCEDURE on AIX and OS/2 535
user labels
  DEBUGGING declarative 537
  LABEL declarative 535
USING phrase
  CALL statement 288
  in Procedure Division header 227
  MERGE statement 349
  on INVOKE statement 338
  SORT statement 411
  subprogram linkage 230

V

V, symbol in PICTURE clause 181
VALUE clause
  condition-name 219
  effect on object-oriented programs 133
  format 217, 219
  level 88 item 140
  NULL 222
  rules for condition-name values 220
  rules for literal values 218
VALUE OF clause
  description 155
  format 144
variable-length tables 176
VARIANCE function 505
VARYING phrase
  PERFORM statement 369
  SEARCH statement 395

W

WHEN phrase
  EVALUATE statement 313
  SEARCH statement 394
WHEN-COMPILED function 506
WHEN-COMPILED special register 19
windowed date field
  See also date field
  definition 59
  expansion before use 165
Windows COBOL language differences
  ACCEPT statement 277
  APPLY WRITE-ONLY 129
  ASSIGN clause 107
  basis-names, library-names, text-names 40
  BLOCK CONTAINS 149
  CLOSE statement 296
  CODE-SET 159
  COMP-5 data item 211
  control characters 142
  environment-name 279
  file handling 129
  file status data-name-8 122
  LABEL RECORDS 154
  LINE SEQUENTIAL file I-O 113
  LOCK MODE clause 120
  MORE-LABELS, GO TO 322
  pointer data item size 214
  RECORD CONTAINS clause 151

Windows COBOL language differences *(continued)*
RECORDING MODE clause 157
RERUN clause 125
RESERVE clause 110
SAME AREA clause 127
SAME SORT AREA clause 128
SAME SORT-MERGE AREA clause 129
SET statement 403
SORT-CONTROL special register 16
SORT-FILE-SIZE under AIX and OS/2 17
SORT-MESSAGE under AIX and OS/2 18
SORT-MODE-SIZE under AIX and OS/2 18
status key value and meaning 271
USE...AFTER...LABEL PROCEDURE 535
WRITE ADVANCING 91, 437
WITH DEBUGGING MODE clause 87, 555
WITH DUPLICATES phrase, SORT statement 409
WITH FOOTING phrase 156
WITH NO ADVANCING phrase 306
WITH NO REWIND phrase, CLOSE statement 297
WITH POINTER phrase
STRING statement 421
UNSTRING statement 431
working storage, levels under AIX and OS/2 403
Working-Storage 133
workstation COBOL language differences
ACCEPT statement 277
APPLY WRITE-ONLY 129
ASSIGN clause 107
basis-names, library-names, text-names 40
BLOCK CONTAINS 149
CLOSE statement 296
CODE-SET 159
COMP-5 data item 211
compiler limits 544
control characters 142
DBCS 2
environment-name 279
file handling 129
file status data-name-8 122
LABEL RECORDS 154
LINE SEQUENTIAL file I-O 113
MORE-LABELS, GO TO 322
pointer data item size 214
RECORD CONTAINS clause 151
RECORDING MODE clause 157
RERUN clause 125
RESERVE clause 110
SAME AREA clause 127
SAME SORT AREA clause 128
workstation COBOL language differences *(continued)*
SAME SORT-MERGE AREA clause 129
SET statement 403
SORT-CONTROL special register 16
SORT-FILE-SIZE under AIX and OS/2 17
SORT-MESSAGE under AIX and OS/2 18
SORT-MODE-SIZE under AIX and OS/2 18
status key value and meaning 271
USE...AFTER...LABEL PROCEDURE 535
WRITE ADVANCING 91, 437
WRITE
ADVANCING under AIX and OS/2 91, 437
WRITE statement
affect on record locking 121
AFTER ADVANCING 438, 442
ALTERNATE RECORD KEY 444
BEFORE ADVANCING 438, 442
description and format 436
END-OF-PAGE phrases 439
FROM identifier phrase 275
sequential files 437

X

X 181
X'00' - X'1F' control characters 142

Y

year 2000 challenge
See date field
year-last date field
See also date field
definition 60
YEAR-TO-YYYY function 508
YEARWINDOW compiler option
century window 61
YEARWINDOW function 509

Z

Z
insertion character 193
symbol in PICTURE clause 181
zero
filling, elementary moves 353
suppression and replacement editing 193
ZERO in sign condition 254

We'd Like to Hear from You

COBOL for OS/390® & VM
COBOL Set for AIX®
VisualAge® COBOL
Language Reference

Publication No. SC26-9046-03

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: COBPUBS@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

COBOL for OS/390® & VM
COBOL Set for AIX®
VisualAge® COBOL
Language Reference

Publication No. SC26-9046-03

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					

May we contact you to discuss your comments? Yes No

Would you like to receive our response by E-Mail?

_____

Your E-mail address

_____ _____
Name Address

_____ _____
Company or Organization

_____ _____
Phone No.

Readers' Comments
SC26-9046-03

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 HHX/H3
PO Box 49023
San Jose, CA 95161-9945

Fold and Tape **Please do not staple** Fold and Tape

SC26-9046-03

Cut or Fold
Along Line

IBM®

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

COBOL for OS/390 & VM

SC26-9046	Language Reference
SC26-9049	Programming Guide
GC26-4764	Compiler and Run-Time Migration Guide
GC26-9045	Installation and Customization under OS/390
SC26-9047	Diagnosis Guide
GC26-9044	Licensed Program Specification
SC09-2137	Debug Tool User's Guide and Reference
GC26-9048	Fact Sheet

COBOL Set for AIX

SC26-9046	Language Reference
SC26-8423	Programming Guide
GC26-8425	Getting Started
GC26-8484	Fact Sheet

VisualAge COBOL

SC26-9046	Language Reference
SC26-9050	Programming Guide
GC26-9051	Getting Started on OS/2
GC26-8944	Getting Started on Windows
SC26-9053	Visual Builder User's Guide
GC26-9052	Fact Sheet

SC26-9 46- 3

Spine information:

IBM IBM COBOL Language Reference