



AIX VS COBOL Compiler/6000

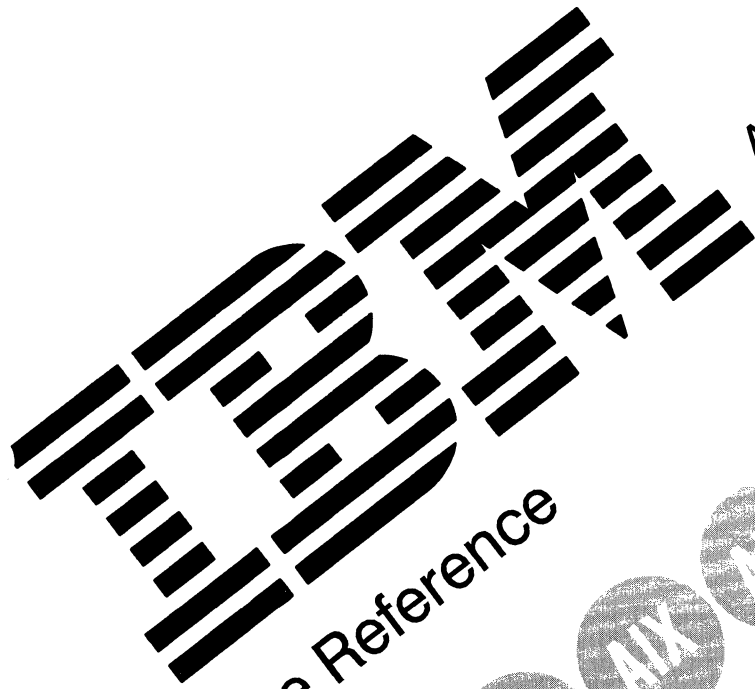
Language Reference



Language Reference

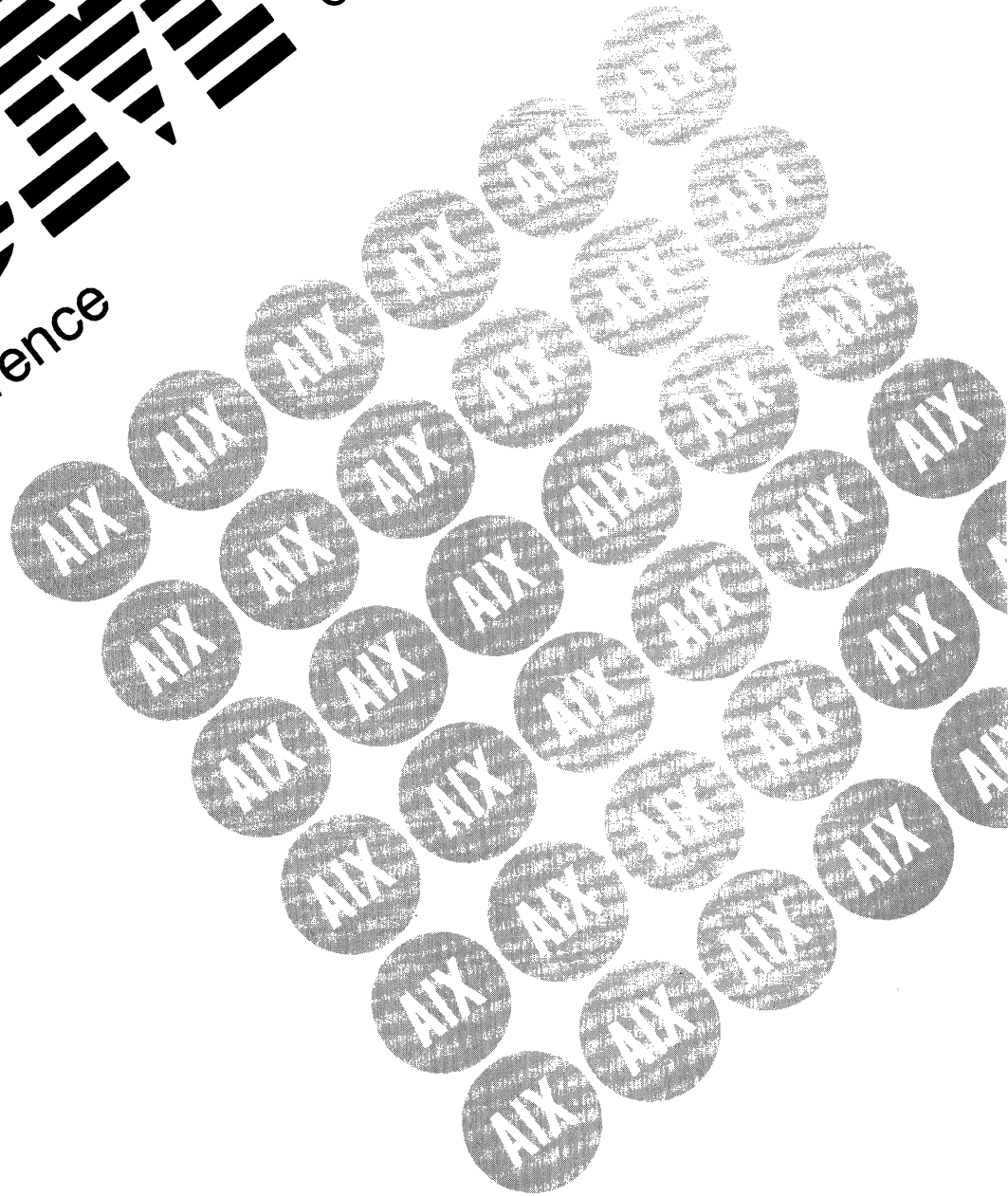
AIX VS COBOL
Compiler/6000





Language Reference

AIX VS COBOL
Compiler/6000



First Edition (March 1990)

This edition of the *Language Reference for IBM AIX VS COBOL Compiler/6000* applies to Version Number 1.1 of the IBM AIX VS COBOL Compiler/6000 Licensed Program and to all subsequent releases of these products until otherwise indicated in new releases or technical newsletters.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed programs. You can use any functionally equivalent program instead.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 997, 11400 Burnet Road, Austin, Texas 78758-3493. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

IBM is a registered trademark of International Business Machines Corporation.

©Copyright International Business Machines Corporation 1987, 1990. All rights reserved.

©Copyright Micro Focus, Ltd. 1987, 1990. All rights reserved.

Notice 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 Corporation.

Trademarks

The following trademarks apply to this book.

- IBM is a registered trademark of International Business Machines Corporation.
- AIX PS/2 VS COBOL is a trademark of International Business Machines Corporation.
- AIX VS COBOL Compiler/6000 is a trademark of International Business Machines Corporation.
- AIX/RT VS COBOL is a trademark of International Business Machines Corporation.
- RT is a registered trademark of International Business Machines Corporation.
- AIX is a trademark of International Business Machines Corporation.
- AIX VS COBOL is a trademark of International Business Machines Corporation.
- OS/VS COBOL and VS COBOL II are trademarks of International Business Machines Corporation.
- VS COBOL is a trademark of Micro Focus.



About This Book

This book discusses the IBM implementation of the VS COBOL language on the AIX Operating System. The book contains language syntax and semantics representing COBOL at the ANSI 85 High Level and ANSI 74 High Level.

Please note that this book does not teach COBOL language programming. You should use this book for reference only.

Who Should Read This Book

This book is intended for persons with some knowledge of COBOL programming concepts and some experience in writing COBOL programs.

This book assumes you know how to use your AIX system. You should be able to log on, create files, edit files, and use various other operating system commands.

How This Book is Organized

Part 1, Introduction and COBOL Concepts

Chapter 1, "Introduction," provides an introduction to the COBOL language elements, book notation style, and formats and rules.

Chapter 2, "COBOL Concepts," describes the language concepts and Identification, Environment, Data, and Procedure Divisions.

Part 2, The Nucleus

Chapter 3, "Introduction to the Nucleus," describes the internal processing of data within the four divisions of a program.

Chapter 4, "Identification Division in the Nucleus," describes the Identification Division of the nucleus.

Chapter 5, "Environment Division in the Nucleus," describes the Environment Division of the nucleus.

Chapter 6, "Data Division," describes the Data Division of the nucleus.

Chapter 7, "Procedure Division in the Nucleus," describes the Procedure Division of the nucleus.

Part 3, File I/O, Source Control, and Inter-Program Communication

Chapter 8, "File Input and Output," describes the use of sequential, relative, and indexed I/O with files.

Chapter 9, "COBOL Source Library," describes editing text, copying text from a source user library file and replacing text in the source program.

Chapter 10, "Listing Control," describes the functions of the list control statement.

Chapter 11, "Interprogram Communication," describes the facility by which a program can communicate with one or more programs.

Part 4, Advanced Features

Chapter 12, "Table-Handling," describes the capability for defining tables of contiguous data items and accessing items within the table.

Chapter 13, “Sort-Merge,” describes the capability of ordering, sorting and merging files.

Chapter 14, “Report Writer,” describes the Report Writer feature, which emphasizes the organization, format, and contents of an output report.

Chapter 15, “Communication,” describes how to access, process, and create messages, and how to communicate with local and remote communication devices.

Chapter 16, “Segmentation,” describes the capability to specify object program overlay requirements.

Chapter 17, “Program Debugging,” describes the procedures for monitoring execution of the object program.

Chapter 18, “Screen-Handling,” describes the enhanced screen handling facilities.

Appendix A, “Ryan-McFarland Syntax Supplement,” provides supplemental syntax information for when you submit your code to the COBOL system and the RM directive is set.

Appendix B, “Data General Syntax Supplement,” provides supplemental syntax information for when you submit your code to the COBOL system and the DG directive is set.

Appendix C, “Microsoft Syntax Supplement,” lists AIX VS COBOL syntax that is compatible with Microsoft COBOL.

Appendix D, “Reserved Word List,” lists the reserved words.

This book also contains an index and a glossary that defines terms used in this publication.

Highlighting

This book uses two sets of highlighting conventions:

- Highlighting within text
- Highlighting within syntax diagrams.

Text

Type styles appearing within the text show the following:

Type Style	Description
Monospace	Examples appear in monospace type.
Bold	Commands, messages, and keywords appear in bold type.
<i>Bold Italic</i>	New terms appear in <i>bold italic</i> type.

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 following symbol indicates the beginning of a statement:



The following symbol indicates that the statement syntax is continued on the next line:

→

The following symbol indicates that a statement is continued from the previous line:

←

The following symbol indicates the end of a statement:

→❖

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

→❖ STATEMENT — required item —→❖

- Optional items appear below the main path.

→❖ STATEMENT ————→❖
 └ optional item ┘

- When you can choose from two or more items, they appear vertically, in a stack. If you **must** choose one of the items, one item of the stack appears on the main path.

→❖ STATEMENT ————→❖
 └ required choice1
 └ required choice2 ┘

If choosing an item is optional, the entire stack appears below the main path.

→❖ STATEMENT ————→❖
 └ optional choice1
 └ optional choice2 ┘

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

→❖ STATEMENT ————→❖
 └ repeatable item ┘

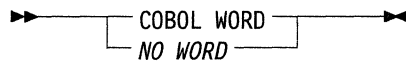
A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

- COBOL key words appear on the main path in uppercase letters. They must be spelled exactly as shown. Optional COBOL words appear below the main path in uppercase letters and are not required.
- Variables appear in all lowercase letters (for example, *parmx*). They represent user supplied names or values.
- Dialects or language extensions appear in italic font with callouts at the right margin identifying the dialects.

→❖ *DIALECT WORD* — *dialect-parm1* —→❖

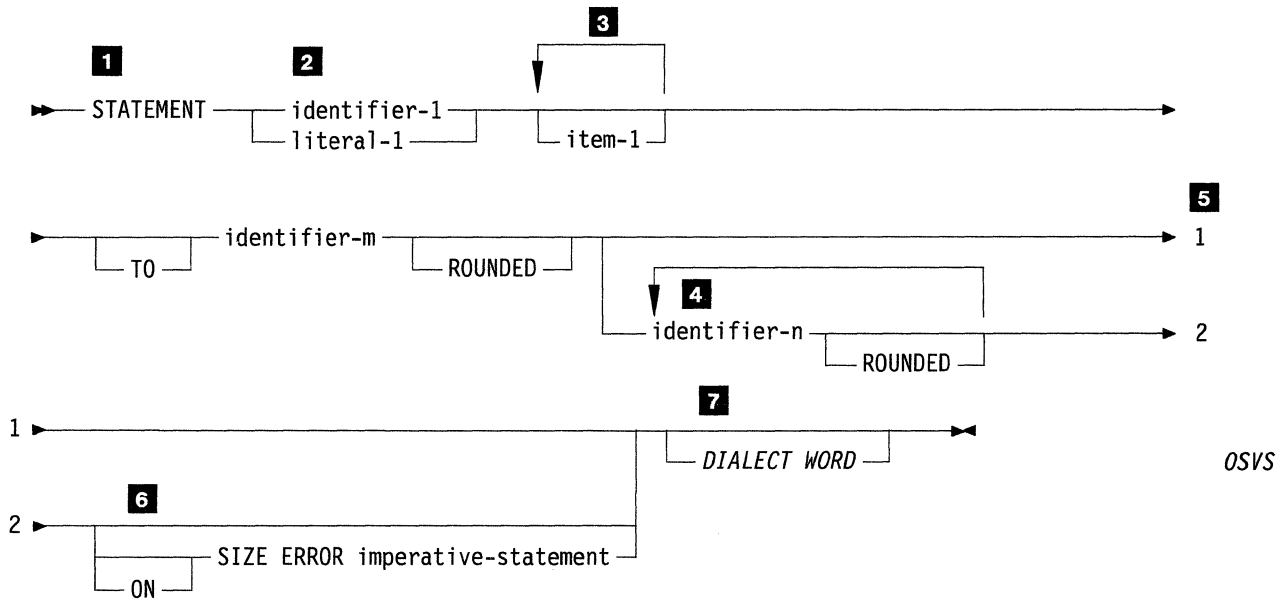
OSVS

- If a dialect allows the omission of a COBOL word or words, a *no word* appears under the COBOL word.

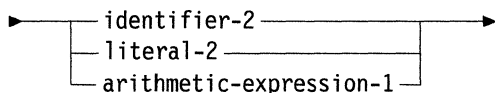


- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, they must be entered as part of the syntax.

The following example shows how the syntax is used:



where item-1 is:



- 1** The STATEMENT key word must be specified and coded as shown.
- 2** This operand is required. Either identifier-1 or literal-1 can be coded.
- 3** The operand item-1 is optional. It can be coded or not, as required by the application. If coded, it may be repeated with each entry separated by one or more blanks. Entry selections allowed for this operand are described at the bottom of the diagram.
- 4** The operand identifier-n is optional. If specified, it can be repeated with one or more blanks separating each entry. Each entry can be assigned the key word ROUNDED.
- 5** In cases where multiple lines must be continued, a number is given at the end of one line and the beginning of the next to show syntax flow.
- 6** The ON SIZE ERROR key word, with associated imperative statement, can be coded after the last identifier-n operand. If coded, SIZE ERROR is required, ON is optional, and it appears below the main path.
- 7** The dialect word is a supported non-ANS85 dialect. The word and the callout both appear in italic font.

The following notations are highlighted to indicate features outside ANSI X3.23-1985:

OSVS For IBM OS/VS COBOL Release 2.4 and earlier.

VSC2 For IBM VS COBOL II Release 2.

MF For Micro Focus extensions.

Related Publications

The AIX VS COBOL Compiler/6000 documentation is available in hardcopy publications only. Softcopy information to support AIX and other licensed programs is provided with the product. The entire AIX library is available as softcopy on a CD-ROM. Refer to the operating system documentation for more detailed information on the various features of AIX. The following hardcopy documentation is also available: *User's Guide for IBM AIX VS COBOL Compiler/6000* describes how to compile and execute AIX VS COBOL programs.

Ordering Additional Copies of This Book

To order additional copies of this publication, use the form number SC23-2177-00.



Contents

PART 1. Introduction and COBOL Concepts

Chapter 1. Introduction	1-1
Contents	1-2
About This Chapter	1-3
IBM AIX VS COBOL Language	1-4
Program Structure	1-7
Manual Format	1-7
Chapter 2. COBOL Concepts	2-1
Contents	2-2
About This Chapter	2-3
Language Concepts	2-4
Character Set	2-4
Language Structure	2-5
Concept Of Computer-Independent Data Description	2-16
Explicit and Implicit Specifications	2-34
Program Structure	2-37
Identification Division	2-38
Environment Division	2-39
Data Division	2-41
Procedure Division	2-44
Reference Format	2-52
Reserved Words	2-55

PART 2. The Nucleus

Chapter 3. Introduction to the Nucleus	3-1
Contents	3-2
About This Chapter	3-3
Function of the Nucleus	3-4
Overall Language	3-4
A COBOL Source Program	3-5
Chapter 4. Identification Division in the Nucleus	4-1
Contents	4-2
About This Chapter	4-3
General Description	4-4
PROGRAM-ID Paragraph	4-6
DATE-COMPILED Paragraph	4-7
REMARKS Paragraph	4-8
Chapter 5. Environment Division in the Nucleus	5-1
Contents	5-2
About This Chapter	5-3
General Description	5-4
Configuration Section	5-4
SOURCE-COMPUTER Paragraph	5-5
OBJECT-COMPUTER Paragraph	5-6
SPECIAL-NAMES Paragraph	5-8
Chapter 6. Data Division	6-1
Contents	6-2
About This Chapter	6-5
WORKING-STORAGE SECTION	6-6

Data Description — Complete Entry Skeleton	6-7
BLANK WHEN ZERO Clause	6-11
Data-Name or FILLER Clause	6-12
JUSTIFIED Clause	6-13
Level Number	6-15
PICTURE Clause	6-18
REDEFINES Clause	6-29
RENAMES Clause	6-32
SIGN Clause	6-35
SYNCHRONIZED Clause	6-37
USAGE Clause	6-39
VALUE Clause	6-41

Chapter 7. Procedure Division in the Nucleus	7-1
Contents	7-2
About This Chapter	7-5
Procedure Division in the Nucleus	7-6
Arithmetic Expressions	7-7
Conditional Expressions	7-9
Common Phrases and General Rules for Statement Formats	7-19
ACCEPT Statement	7-22
ADD Statement	7-24
ALTER Statement	7-27
COMPUTE Statement	7-29
CONTINUE Statement	7-31
DISPLAY Statement	7-32
DIVIDE Statement	7-34
ENTER Statement	7-38
EVALUATE Statement	7-39
EXAMINE Statement	7-43
EXEC(UTE) Statement	7-45
EXHIBIT Statement	7-46
EXIT Statement	7-48
GO TO Statement	7-50
IF Statement	7-52
INITIALIZE Statement	7-54
INSPECT Statement	7-57
MOVE Statement	7-65
MULTIPLY Statement	7-69
ON Statement	7-71
PERFORM Statement	7-73
SET Statement	7-84
STOP Statement	7-86
STRING Statement	7-87
SUBTRACT Statement	7-91
TRANSFORM Statement	7-94
UNSTRING Statement	7-96

PART 3. File I/O, Source Control, and Inter-Program Communication

Chapter 8. File Input and Output	8-1
Contents	8-2
About This Chapter	8-5
Introduction	8-6
Sharing Files on Multiuser Systems	8-12
Environment Division for File Input and Output	8-16
INPUT-OUTPUT SECTION	8-16
FILE-CONTROL Paragraph	8-17
FILE-CONTROL Entry	8-18
I-O Control Paragraph	8-29
Data Division for File Input and Output	8-34
BLOCK CONTAINS Clause	8-39

CODE-SET Clause	8-40
DATA RECORDS Clause	8-42
LABEL RECORDS Clause	8-43
LINAGE Clause	8-44
RECORD Clause	8-47
RECORDING MODE Clause	8-50
VALUE OF Clause	8-51
Procedure Division for File Input and Output	8-53
CLOSE Statement	8-53
COMMIT Statement	8-58
DELETE Statement	8-59
OPEN Statement	8-62
READ Statement	8-68
REWRITE Statement	8-75
START Statement	8-79
UNLOCK Statement	8-85
USE Statement	8-86
WRITE Statement	8-89
Chapter 9. COBOL Source Library	9-1
Contents	9-2
About This Chapter	9-3
Introduction	9-4
COPY Statement	9-5
REPLACE Statement	9-10
Chapter 10. Listing Control	10-1
Contents	10-2
About This Chapter	10-3
SKIP1, SKIP2, and SKIP3 Statements	10-4
EJECT Statement	10-5
TITLE Statement	10-6
Chapter 11. Interprogram Communication	11-1
Contents	11-2
About This Chapter	11-5
Introduction	11-6
Language Concepts	11-6
Nested Source Programs	11-10
END PROGRAM Header	11-13
Identification Division in the Interprogram Communication Module	11-15
Data Division in the Interprogram Communication Module	11-17
LINKAGE SECTION	11-17
File Description Entry in the Interprogram Communication Module	11-19
Data Description Entry in the Interprogram Communication Module	11-25
Report Description Entry in the Interprogram Communication Module	11-28
EXTERNAL Clause	11-30
GLOBAL Clause	11-31
Procedure Division in the Interprogram Communication Module	11-33
CALL Statement	11-36
CANCEL Statement	11-43
CHAIN Statement	11-45
ENTRY Statement	11-47
EXIT PROGRAM Statement	11-50
GOBACK Statement	11-51
USE Statement	11-52
USE BEFORE REPORTING Statement	11-53
PART 4. Advanced Features	
Chapter 12. Table-Handling	12-1

Contents	12-2
About This Chapter	12-3
Introduction	12-4
Data Division in the Table-Handling Module	12-5
OCCURS Clause	12-5
USAGE IS INDEX Clause	12-12
Procedure Division in the Table-Handling Module	12-13
SEARCH Statement	12-14
SET Statement	12-19
Table-Handling Sample Program	12-21
Chapter 13. Sort-Merge	13-1
Contents	13-2
About This Chapter	13-3
Introduction	13-4
Environment Division in the Sort-Merge Module—Input-Output Section	13-4
FILE-CONTROL Paragraph	13-5
FILE-CONTROL Entry	13-6
I-O-CONTROL Paragraph	13-8
Data Division in the Sort-Merge Module	13-10
Sort-Merge File Description - Complete Entry Skeleton	13-10
DATA RECORDS Clause	13-12
RECORD CONTAINS Clause	13-13
Procedure Division in the Sort-Merge Module	13-14
MERGE Statement	13-14
RELEASE Statement	13-18
RETURN Statement	13-19
SORT Statement	13-21
Sort-Merge Sample Program	13-26
Chapter 14. Report Writer	14-1
Contents	14-2
About This Chapter	14-5
Introduction	14-6
REPORT SECTION	14-6
Report Structure	14-6
Environment Division in the Report Writer Module	14-10
INPUT-OUTPUT Section	14-10
FILE-CONTROL Paragraph	14-10
I-O-CONTROL Paragraph	14-10
Data Division in the Report Writer Module	14-11
File Description Entry	14-11
REPORT Clause	14-14
REPORT SECTION	14-16
CODE Clause	14-19
CONTROL Clause	14-20
PAGE Clause	14-22
Example 1	14-25
Example 2	14-26
Example 3	14-27
Report Group Description Entry	14-28
COLUMN NUMBER Clause	14-45
Data-Name	14-46
GROUP INDICATE Clause	14-47
Level-Number	14-48
LINE NUMBER Clause	14-49
NEXT GROUP Clause	14-51
SIGN Clause	14-53
SOURCE Clause	14-55
SUM Clause	14-56
TYPE Clause	14-59
USAGE Clause	14-63

VALUE Clause	14-64
Procedure Division in the Report Writer Module	14-66
CLOSE Statement	14-67
GENERATE Statement	14-68
INITIATE Statement	14-71
OPEN Statement	14-72
SUPPRESS Statement	14-73
TERMINATE Statement	14-74
USE BEFORE REPORTING Statement	14-76
Report Writer Sample Program	14-77
Chapter 15. Communication	15-1
Contents	15-2
About This Chapter	15-3
Introduction	15-4
Data Division in the Communication Module	15-4
Procedure Division in the Communication Module	15-19
ACCEPT MESSAGE COUNT Statement	15-20
DISABLE Statement	15-21
ENABLE Statement	15-23
PURGE Statement	15-25
RECEIVE Statement	15-26
SEND Statement	15-29
Communication Sample Program	15-33
Chapter 16. Segmentation	16-1
Contents	16-2
About This Chapter	16-3
Introduction	16-4
General Description of Segmentation	16-4
Structure of Program Segments	16-6
Restrictions on Program Flow	16-7
Chapter 17. Program Debugging	17-1
Contents	17-2
About This Chapter	17-3
Introduction	17-4
Environment Division in COBOL Debug	17-5
WITH DEBUGGING MODE Clause	17-5
Procedure Division in COBOL Debug	17-6
READY TRACE Statement	17-6
RESET TRACE Statement	17-7
USE FOR DEBUGGING Statement	17-8
Debugging Lines	17-13
Debugging Facilities Sample Program	17-14
Chapter 18. Screen-Handling	18-1
Contents	18-2
About This Chapter	18-5
Introduction	18-6
Environment Division in the Screen-Handling Module	18-8
SPECIAL-NAMES Paragraph	18-9
CONSOLE IS CRT Clause	18-10
CURSOR IS Clause	18-11
CRT STATUS Clause	18-12
Data Division in the Screen-Handling Module	18-14
SCREEN SECTION	18-14
Screen Description — Complete Entry Skeleton	18-15
AUTO Clause	18-21
BACKGROUND-COLOR Clause	18-22
BELL Clause	18-24
BLANK Clause	18-25

BLANK WHEN ZERO Clause	18-26
BLINK Clause	18-27
COLUMN Clause	18-28
BACKGROUND-COLOR Clause	18-30
FULL Clause	18-32
GRID Clause	18-34
HIGHLIGHT Clause	18-35
JUSTIFIED Clause	18-36
LEFTLINE Clause	18-37
LINE Clause	18-38
OCCURS Clause	18-40
OVERLINE Clause	18-42
PICTURE Clause	18-43
PROMPT Clause	18-45
REQUIRED Clause	18-46
REVERSE-VIDEO Clause	18-47
SECURE Clause	18-48
SIGN Clause	18-49
SIZE Clause	18-50
UNDERLINE Clause	18-51
VALUE Clause	18-52
ZERO-FILL Clause	18-53
Procedure Division	18-54
ACCEPT Statement	18-55
DISPLAY Statement	18-61
Appendix A. Ryan-McFarland Syntax Supplement	A-1
Introduction	A-3
Reserved Words	A-3
Identification Division - The PROGRAM-ID Paragraph	A-3
Environment Division	A-3
Data Division	A-6
Procedure Division	A-8
Appendix B. Data General Syntax Supplement	B-1
Introduction	B-3
Long User-Defined Names	B-3
Environment Division	B-4
Data Division	B-6
Procedure Division	B-7
Appendix C. Microsoft Syntax Supplement	C-1
Introduction	C-3
Compatibility with Microsoft COBOL	C-3
Dialect Controlling Directives	C-3
Summary of Syntactic Differences	C-3
Problem Determination	C-8
Appendix D. Reserved Word List	D-1
Introduction	D-3
Appendix E. Obsolete Language Elements	E-1
Introduction	E-3
List of Obsolete Language Elements	E-3
Glossary	G-1
Index	X-1

Figures

1-1.	A Sample COBOL Program Showing Source Format	1-10
2-1.	Example of Level Numbers in Group Descriptions	2-17
2-2.	Example of Level Numbers Representing a Data Hierarchy	2-17
2-3.	Storage Allocation	2-18
2-4.	Sample Computer Storage Allocation	2-23
2-5.	Number Storage	2-27
2-6.	Reference Format for a COBOL Source Line	2-52
6-1.	PICTURE Character Precedence Chart	6-28
7-1.	Relational Operators	7-11
7-2.	INSPECT Statement and the Execution Result	7-63
7-3.	The VARYING Option of a PERFORM Statement with the TEST BEFORE Phrase Having One Condition	7-78
7-4.	The VARYING Option of a PERFORM Statement with the TEST BEFORE Phrase Having Two Conditions	7-79
7-5.	The VARYING Option of a PERFORM Statement with the TEST AFTER Phrase Having One Condition	7-80
7-6.	The VARYING Option of a PERFORM Statement with a TEST AFTER Phrase Having Two Conditions	7-81
8-1.	Valid Combinations of File Status Keys 1 and 2	8-12
8-2.	Relationship of CLOSE Statement with File Category	8-57
12-1.	Flowchart of SEARCH Operation Containing Two WHEN Phrases	12-18
14-1.	Illustration of PAGE Clause Ranges	14-24
14-2.	Values Assumed for Omitted PAGE Clause Options	14-24
14-3.	Report Heading Group Presentation Rules Table	14-36
14-4.	Page Heading Group Presentation Rules Table	14-37
14-5.	Body Group Presentation Rules	14-39
14-6.	Report Footing Presentation Rules	14-44
15-1.	Communication Status Key Condition	15-8
15-2.	Error Key Values	15-9
18-1.	Permitted Use Of Options	18-7

Tables

2-1.	Figurative Constant Values and the Reserved Words	2-12
2-2.	Special Registers, Implicit Data Description Picture, and Usage	2-13
2-3.	Data Levels, Classes, and Categories	2-19
2-4.	Incorporation of Sign Data into the Requisite Digit	2-21
2-5.	COMP(UTATIONAL) Format Data Item Character-Position (Byte) Storage Assignment	2-22
2-6.	Change of Results Due to TRUNC Directive	2-24
2-7.	Binary-Coded Decimal Form	2-25
2-8.	COMPUTATIONAL-3 Sign Digit Representation	2-26
2-9.	Numeric Data Storage for the COMP(UTATIONAL)-3 or PACKED-DECIMAL PICTURE Clause	2-26
2-10.	Explicit Scope Terminators	2-36
5-1.	Function-Name Reference	5-14
6-1.	Editing Types for Data Categories	6-23
6-2.	Editing Symbols in PICTURE Character Strings	6-24
7-1.	Combination of Symbols in Arithmetic Expressions	7-8
7-2.	Combinations of Conditions, Logical Operators, and Parentheses	7-17
7-3.	MOVE Statement Data Categories	7-68
8-1.	Default Locking for Sequential Files	8-14
8-2.	Default Locking for Relative and Indexed Files	8-15
8-3.	AFTER POSITIONING Phrase with identifier-2	8-95
8-4.	AFTER POSITIONING Phrase with integer-1	8-95
12-1.	Storage Layout for Table-Three	12-9
12-2.	SET Statement Valid Operand Combinations	12-20
13-1.	Status Key Combinations	13-7
14-1.	Page Regions Established by the Page Clause	14-25
14-2.	Permissible Clause Combinations in Format 3 Entries	14-32
14-3.	Page Footing Presentation Rules	14-41
18-1.	Valid Combinations of CRT Status Keys 1 and 2	18-13
A-1.	Mapping of File I-O Status Codes	A-11
D-1.	Reserved Words	D-4

PART 1. Introduction and COBOL Concepts

Chapter 1. Introduction

Contents

About This Chapter	1-3
IBM AIX VS COBOL Language	1-4
Supported Language Elements of IBM OS/VS COBOL	1-4
Supported Elements of IBM VS COBOL II	1-4
Supported Micro Focus COBOL Enhancements	1-5
Supported Double-Byte Character Set Features (DBCS)	1-5
Program Structure	1-7
Manual Format	1-7
General Format	1-7
Syntax Rules	1-8
General Rules	1-8
Elements	1-8
Source Format	1-8
Sequence Number	1-8
Indicator Area	1-8
Areas A and B	1-9

About This Chapter

This chapter describes the following:

- The AIX VS COBOL language coverage
- The formats and rules used in this manual
- The source format of COBOL source records.

IBM AIX VS COBOL Language

Common Business Oriented Language, COBOL, is the programming language most widely used in commercial and administrative data processing.

The AIX VS COBOL language is a superset of ANSI COBOL 1985 HIGH as specified in American National Standard Programming Language COBOL ANSI X3.23-1985. Language extensions include:

- Some of the more commonly used extensions of IBM OS/VS COBOL (Release 2.4 and earlier)
- Most of the language constructs of IBM VS COBOL II (Release 2), December, 1986
- Additional extensions unique to this implementation of COBOL such as:
 - Micro Focus specific extensions
 - Ryan McFarland syntax
 - Data General syntax
 - Microsoft syntax.

Combinations of these extensions are permissible in program source. In addition, a flagging option to identify the various language extensions in COBOL source programs ensures that programs can be verified as being valid for any given one of the language specifications.

AIX VS COBOL Compiler/6000 allows programmers to write SAA-conforming programs according to the *SAA CPI COBOL Reference*, SC26-4354-1.

AIX VS COBOL Compiler/6000 is source code compatible with AIX/RT VS COBOL and with AIX PS/2 VS COBOL.

AIX VS COBOL fully supports the ANSI optional modules Segmentation, Report Writer, and Debug. The syntax for the ANSI optional module Communications is accepted, but Communications is not supported at run time.

Supported Language Elements of IBM OS/VS COBOL

The elements of OS/VS COBOL that are fully supported include:

- The EXAMINE, EXHIBIT, TRANSFORM, ON, and GOBACK procedural statements
- The RETURN-CODE, CURRENT-DATE, TIME-OF-DAY, TALLY, and WHEN-COMPILED special registers
- Language Level 1 COPY statement and COPY...SUPPRESS feature
- 2, 4, and 8 byte COMPUTATIONAL and SYNCHRONIZED numeric data memory allocation option
- The ENTRY...USING statement mechanism
- The REMARKS paragraph and EJECT and SKIP statements.

Many other language features are supported either fully or at the documentary level.

Supported Elements of IBM VS COBOL II

Some supported elements of IBM VS COBOL II not already listed above are:

- Nested COPY statements
- LENGTH special register
- TITLE statement

- USAGE POINTER
- ADDRESS special register.

Supported Micro Focus COBOL Enhancements

A screen-handling module, comprising a SCREEN SECTION and additional formats of ACCEPT and DISPLAY statements, which enables the user to specify exact location of fields on a display screen, accept data entered at specified positions, display literal text at specified positions, define display screen attributes, and control console features.

Enhanced file I-O with an additional ORGANIZATION to handle text files efficiently and to handle optional definition of file names either within a program source (data variable or literal) or at run time from operating system environment variables. File sharing features and record locking are also provided.

Supported Double-Byte Character Set Features (DBCS)

AIX VS COBOL supports the use of Double-Byte Character Set (DBCS) features as specified in the *SAA CPI COBOL Reference*. In this manual, references to DBCS supported features will be enclosed in a framing box, as follows:

DBCS Support
End of DBCS Support

Note: To use the DBCS-supporting implementation of AIX VS COBOL, you must have installed the DBCS version of the AIX VS COBOL compiler, and you must use the DBCS compiler option on your compilations. See the *User's Guide* for more information on installation and options.

Below is a summary of supported DBCS features. These include all DBCS features supported in the *SAA CPI COBOL Reference*. Features specified in SAA have page references to the *SAA CPI COBOL Reference* (SC26-4354-1).

SAA	Supported Feature
18	Summary of DBCS elements supported
20	DBCS characters allowed in character strings
22	DBCS character strings allowed in literals and comments
26	There is a DBCS type of literal; mixed strings
26	SBCS and DBCS characters can be mixed in nonnumeric literals
27	Specification for G“ ” DBCS literal
28	DBCS characters allowed in comments
34	DBCS literals cannot be continued
44	DBCS character strings allowed in comments in ID division
54	CURRENCY SIGN clause cannot use uppercase G
63	RECORD KEY clause in Environment Division may use DBCS data item
78	DBCS class and category added to chart of data categories
93	BLANK WHEN ZERO clause not allowed for DBCS items

-
- 94 JUSTIFIED RIGHT may be used for DBCS items
 - 94 OCCURS clause may be specified for a DBCS item
 - 95 ASC/DESC KEY phrase may use DBCS item in OCCURS clause in SEARCH ALL
 - 102 Lowercase g is equivalent to uppercase G for DBCS PIC strings
 - 103 PICTURE clause description for G, DBCS B
 - 106 PICTURE symbol sequence chart
 - 106 G may appear more than once in one PIC character string
 - 106 G may appear alone in the PIC character string
 - 108 DBCS and DBCS-edited are additional data categories
 - 110 PICTURE string description for DBCS and DBCS-edited item
 - 111 Describes insertion symbols allowed for DBCS items
 - 119 RENAME clause in Data Division may specify DBCS items
 - 122 SYNCHRONIZED clause is ignored for DBCS items
 - 124 DISPLAY-1 is added as a USAGE type
 - 126 DISPLAY-1 usage phrase defines a DBCS item
 - 129 VALUE clause associated with a DBCS item must contain a DBCS literal
 - 131 DBCS literals allowed in VALUE clauses
 - 132 Relation tests allowed for DBCS items
 - 134 DBCS items allowed in PROCEDURE DIVISION USING clause
 - 143 Relational operators can be used with DBCS items
 - 147 Rules for comparing DBCS operands are same as nonnumeric operands
 - 180 CALL USING phrase may use DBCS items
 - 201 EVALUATE statement may use DBCS items and literals
 - 211 INITIALIZE...REPLACING may use DBCS as a category
 - 216 INSPECT statement may use DBCS items and literals
 - 227 MOVE statement may use DBCS items and literals
 - 228 MOVE statement semantics for DBCS items and literals
 - 229 MOVE statement conversion semantics for DBCS items
 - 230 MOVE statement table of sending/receiving categories
 - 253 READ statement: INTO phrase may use DBCS items
 - 254 READ statement: KEY IS phrase may use DBCS items
 - 263 REWRITE statement: FROM identifier may be a DBCS item
 - 267 SEARCH statement: identifier-1 may be DBCS item
 - 268 SEARCH statement: WHEN phrase may use DBCS relations, conditional names
 - 270 SEARCH statement may include DBCS items for ASC/DESC KEY items
 - 280 START data-name may be a DBCS item
 - 286 STRING statement may use DBCS items and literals
 - 293 UNSTRING statement can operate on DBCS item
 - 293 UNSTRING statement: DELIMITED BY phrase may use DBCS items
 - 294 UNSTRING statement: INTO phrase may use DBCS items

-
- 295 UNSTRING statement: POINTER phrase may use DBCS items
 - 302 WRITE statement may use DBCS items
 - 317 TITLE statement may use DBCS literals
 - 329 DBCS, DISPLAY-1 included in Reserved Word list
 - 341 Glossary defines DBCS
 - 357 Index entry for DBCS
 - 357 Index entry for DISPLAY-1
 - 359 Index entry for G symbol in PICTURE clause

In addition to SAA DBCS support, AIX VS COBOL provides DBCS support for the following features:

- Condition tests (level 88 items may be DBCS items)
- ENTRY...USING may use DBCS items
- COPY...REPLACING may use DBCS items.

Program Structure

A COBOL program consists of four divisions:

1. Identification Division — An identification of the program
2. Environment Division — A description of the equipment to be used to compile and run the program
3. Data Division — A description of the data to be processed
4. Procedure Division — A set of procedures to specify the operations to be performed on the data.

Each division is divided into sections, which are further divided into paragraphs, which in turn are made up of sentences.

Within these subdivisions of a COBOL program, further subdivisions exist as clauses and statements. A clause is an ordered set of COBOL elements that specify an attribute of an entry, and a statement is a combination of elements in the Procedure Division that includes a COBOL verb and constitutes a program instruction.

Manual Format

This section describes the format of this manual.

General Format

A general format is the specific arrangement of the elements of a clause or a statement. Throughout this document a format is shown adjacent to information defining the clause or statement. When more than one specific arrangement is permitted, the general format is separated into numbered formats. Clauses must be written in the sequence given in the general formats. (Clauses that are optional must appear in the sequence shown if they are used.) In certain cases, stated explicitly in the rules associated with a given format, the clauses may appear in sequences other than those shown. Applications, requirements, or restrictions are shown as rules.

Syntax Rules

Syntax rules define or clarify the order in which words or elements are arranged to form larger elements, such as phrases, clauses, or statements. Syntax rules also impose restrictions on individual words or elements.

These rules are used to define or clarify how the statement must be written. This includes the order of the elements of the statement and restrictions on what each element may represent.

General Rules

A general rule defines or clarifies the meaning, or relationship of meanings, of an element or set of elements. It is used to define or clarify the semantics of the statements, and the effect it has on execution or production of intermediate code.

Elements

Elements which make up a clause or a statement consist of uppercase words, lowercase words, level numbers, brackets, braces, connectives, and special characters (see Chapter 2, "COBOL Concepts").

Source Format

The COBOL source format divides each COBOL source record into 72 columns. These columns are used in the following ways:

Columns 1-6	Sequence number
Column 7	Indicator area
Columns 8-11	Area A
Columns 12-72	Area B

See "Reference Format" on page 2-52 for more details.

Sequence Number

A sequence number of six digits may be used to identify each source program line. If column 1 contains an asterisk (*), or columns 1 and 2 contain a form feed character followed by an asterisk, the entire line is ignored by the compiler and does not appear in the list file. This facility allows list files to be used as source files.

Indicator Area

An asterisk (*) in the indicator area marks the line as documentary comment only. Such a comment line can appear anywhere in the program after the Identification Division header. Any characters from the ASCII character set can be included in area A and area B of the line.

A stroke (/) in the indicator area acts as a comment line above but causes the page to eject before printing the comment.

A letter D in the indicator area represents a debugging line. Areas A and B may contain any valid COBOL sentence.

A hyphen (-) in the indicator area represents a continuation of the previous line without spaces or the continuation of a nonnumeric literal (see Chapter 2, "COBOL Concepts").

Areas A and B

Section names and paragraph names begin in area A and are followed by a period and a space. Level indicators FD, SD, CD, and 01, 66, 77, 78, and 88 begin in area A and are followed in area B by the appropriate file and record description.

MF

No rules regarding area A and area B are enforced except in relation to comment entries in the Identification Division.

MF

More than one sentence is permitted in each source record. The source format of a typical program is illustrated in Figure 1-1.

```
1*****
2* COBOL INVADERS CALLING PROGRAM      *
3*      VERSION 1.1                    *
4*      2/8/84                         *
5* COPYRIGHT (C) MICRO FOCUS 1984     *
6*****
7 SPECIAL-NAMES.
8 CONSOLE IS CRT.
9 DATA DIVISION.
10 WORKING-STORAGE SECTION.
11 01 GET-CHARACTER  PIC X    VALUE X"83".
12 01 CHARACTER-FOUND  PIC 99  COMP.
13 01 SCAN-KEYBOARD  PIC X    VALUE X"D9".
14 01 SCAN-RESULT    PIC 99  COMP.
15 COPY "READY.DDS".
.
.                                     (COPY FILE INCLUDED HERE.)
.
35 01 SCREEN-IO-PARAMETERS.
36   03 SCREEN-IO    PIC X    VALUE X"87".
37   03 WRITE-TEXT   PIC 99  COMP VALUE 1.
38   03 READ-TEXT    PIC 99  COMP VALUE 0.
39   03 WRITE-ATTRIB PIC 99  COMP VALUE 3.
40 01 FORM-PARAMS.
41   03 IO-LENGTH    PIC 9(4) COMP VALUE 1094.
42   03 SCREEN-OFFSET PIC 9(4) COMP VALUE 1.
43   03 BUFFER-OFFSET PIC 9(4) COMP VALUE 1.
44 01 SCORE-PARAMS.
45   03 S-IO-LENGTH  PIC 9(4) COMP VALUE 23.
46   03 S-SCREEN-OFFSET PIC 9(4) COMP VALUE 1841.
47   03 S-BUFFER-OFFSET PIC 9(4) COMP VALUE 1.
48 01 SCORE-TEXT.
49   03 S-TEXT-1     PIC X(12) VALUE "LAST SCORE:".
50   03 SCORE        PIC 9(4).
51   03 S-TEXT-2     PIC X(7)  VALUE "POINTS".
52 01 SCREEN-ATTR-1 PIC X(1094).
53 01 CURSOR-POSITION.
54   03 CURSOR-LINE  PIC 99  COMP VALUE 25.
55   03 CURSOR-CHAR  PIC 99  COMP VALUE 00.
56 01 DUMMY          PIC 99  COMP.
57 01 MOVE-CURSOR-ROUTINE PIC X    VALUE X"E6".
58 PROCEDURE DIVISION.
59 START-UP.
```

Figure 1-1 (Part 1 of 2). A Sample COBOL Program Showing Source Format

```

60 DISPLAY SPACE
61     CALL MOVE-CURSOR-ROUTINE USING DUMMY, CURSOR-POSITION
62     CALL SCREEN-IO USING WRITE-TEXT, FORM-PARAMS, READY-00
63     CALL SCREEN-IO USING WRITE-TEXT, SCORE-PARAMS, SCORE-TEXT.
64 LOOP.
65     MOVE ALL "?" TO SCREEN-ATTR-1
66     CALL SCREEN-IO USING WRITE-ATTRIB, FORM-PARAMS, SCREEN-ATTR-1
67     PERFORM DELAY-LOOP 30 TIMES
68*****HAS A KEY BEEN PRESSED?
69     CALL SCAN-KEYBOARD USING SCAN-RESULT
70     IF SCAN-RESULT = 1
71     GO TO CALL-INVADERS.
72     MOVE ALL " " TO SCREEN-ATTR-1
73     CALL SCREEN-IO USING WRITE-ATTRIB, FORM-PARAMS, SCREEN-ATTR-1
74     PERFORM DELAY-LOOP 30 TIMES
75*****HAS A KEY BEEN PRESSED?
76     CALL SCAN-KEYBOARD USING SCAN-RESULT
77     IF SCAN-RESULT = 1
78     GO TO CALL-INVADERS.
79     GO TO LOOP.
80 DELAY-LOOP.
81 CALL INVADERS.
82*****WHICH KEY WAS PRESSED?
83     CALL GET-CHARACTER USING CHARACTER-FOUND
84     IF CHARACTER-FOUND = 27
85     STOP RUN.
86     IF CHARACTER-FOUND = 32
87     CALL "INV-SUB" USING SCORE
88     GO TO START-UP.
89     GO TO LOOP.

```

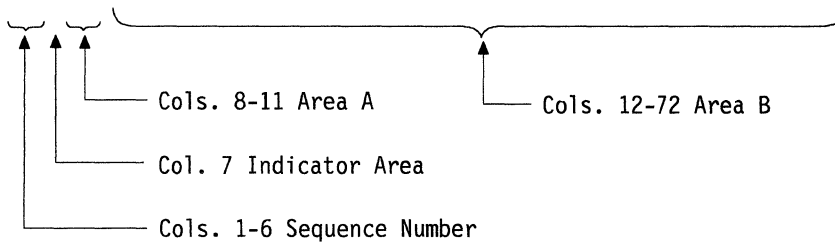


Figure 1-1 (Part 2 of 2). A Sample COBOL Program Showing Source Format



Chapter 2. COBOL Concepts

Contents

About This Chapter	2-3
Language Concepts	2-4
Character Set	2-4
Language Structure	2-5
Separators	2-5
Character-Strings	2-6
COBOL Words	2-6
Literals	2-9
PICTURE Character-Strings	2-15
Comment-Entries	2-15
Concept Of Computer-Independent Data Description	2-16
Concept Of Levels	2-16
Concepts of Classes of Data	2-18
Algebraic Signs	2-19
Standard Alignment Rules	2-19
Item Alignment for Increased Object Code Efficiency	2-20
Selection of Character Representation and Radix	2-20
DISPLAY Format	2-21
COMPUTATIONAL, COMP, BINARY, <i>COMPUTATIONAL-4</i> , or <i>COMP-4</i> Format	2-21
<i>COMPUTATIONAL-3 COMP-3</i> PACKED-DECIMAL Format	2-25
<i>COMPUTATIONAL-5 Or COMP-5 Format</i>	2-27
<i>COMPUTATIONAL-X Or COMP-X Format</i>	2-28
<i>POINTER Format</i>	2-28
Uniqueness of Reference	2-28
Explicit and Implicit Specifications	2-34
Explicit And Implicit Procedure Division References	2-34
Explicit and Implicit Transfers of Control	2-34
Explicit and Implicit Attributes	2-35
Explicit and Implicit Scope Terminators	2-36
Program Structure	2-37
Optional Division, Section, and Paragraph Headings	2-37
Identification Division	2-38
Organization	2-38
Structure	2-38
General Format	2-38
Environment Division	2-39
Organization	2-39
Structure	2-39
General Format	2-40
Data Division	2-41
Data Division Organization	2-41
General Format	2-42
Procedure Division	2-44
Statements and Sentences	2-46
Reference Format	2-52
Reference Format Representation	2-52
Division, Section, and Paragraph Formats	2-54
Data Division Entries	2-54
Reserved Words	2-55

About This Chapter

This chapter describes the basic COBOL concepts. The following aspects are covered:

- Basic language elements, such as character set, language structure, and data description concepts
- Program structure
- Identification Division
- Environment Division
- Data Division
- Procedure Division.

Language Concepts

This section describes the following:

- Character Sets
- Language Structures
- Separators
- Characters
- Strings
- COBOL Words
- Literals
- Figurative Constant Values
- Constant Names
- Special Registers
- PICTURE Character-Strings
- Comment Entries.

Character Set

The most basic and indivisible unit of the language is the character. The set of characters used to form IBM AIX VS COBOL character-strings and separators includes the letters of the alphabet, digits, and special characters. The character set consists of the following characters:

0 to 9	Digits
A to Z	Uppercase letters
a to z	Lowercase letters
Space	
+	Plus sign
-	Minus sign or hyphen
*	Asterisk
/	Oblique stroke/slash
=	Equal sign
\$	Dollar sign
.	Period (full stop) or decimal point
,	Comma or decimal point
;	Semicolon
"	Quotation mark
'	<i>Apostrophe</i>
(Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol
:	Colon

OSVS VSC2

Lowercase letters may be used in character-strings. Each lowercase letter is equivalent to the corresponding uppercase letter except when used in nonnumeric literals.

The AIX VS COBOL language is restricted to the preceding character set, but the content of nonnumeric literals, comment lines, comment entries, and data may include any ASCII characters. For more information refer to Appendix D, "Reserved Word List."

Characters from the Double-Byte Character Set (DBCS) are valid characters in certain COBOL character-strings.

Lowercase g is equivalent to uppercase G in DBCS PICTURE strings.

Language Structure

The individual characters of the language are concatenated to form character-strings and separators. A separator may be concatenated with another separator or with a character-string. A character-string may only be concatenated with a separator. The concatenation of character-strings and separators forms the text of a source program.

Separators

A separator is a string of one or more punctuation characters. The rules for forming separators are as follows:

1. The space character is a separator. More than one space can be used as a single separator. All spaces immediately following the comma, semicolon, or period separators are considered part of that separator and are not considered to be the space separator.
2. Commas and semicolons, when followed immediately by a space are separators that can be used anywhere the separator space is used. However, the comma is always used in a PICTURE character-string.
3. The period character, when followed by a space, is a separator. The period character must be used only to indicate the end of a sentence, or as shown in formats.
4. The right and left parenthetical symbols are separators. Parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, indexes, arithmetic expressions, or conditions.
5. The quotation mark character (") is a separator. An opening quotation mark must be preceded immediately by a space or left parenthesis. A closing quotation mark must be followed immediately by a space, comma, semicolon, period, or right parenthesis separator.

Quotation marks may appear only in balanced pairs delimiting nonnumeric literals, except when the literal is continued. Refer to "Continuation of Lines" on page 2-53.

6. *The apostrophe character may replace the quotation mark character in a OSVS VSC2 program.*

Both the quotation mark and the apostrophe may appear within the same program. *MF*

7. Pseudo-text delimiters are separators. An opening pseudo-text delimiter must be preceded immediately by a space. A closing pseudo-text delimiter must be followed immediately by a space, comma, semicolon, or period separator.

Pseudo-text delimiters may appear only in balanced pairs delimiting pseudo-text. Refer to Chapter 8, "File Input and Output."

-
8. The separator space can immediately precede all separators except the following:
 - a. As specified by reference format. Refer to “Reference Format” on page 3-4.
 - b. The separator closing quotation mark. In this case, a preceding space is considered to be part of the nonnumeric literal and not as a separator.
 - c. The opening pseudo-text delimiter, where the preceding space is required.
 9. The comma and semicolon separators may be used anywhere the separator space is used in the formats. In the source program, comma and semicolon are interchangeable.
 10. The separator space may follow immediately any separator except the opening quotation mark. In this case, a following space is considered as part of the nonnumeric literal and not as a separator.

A punctuation character which appears as part of the specification of a PICTURE character-string (refer to Chapter 3, “Introduction to the Nucleus”) or numeric literal is not considered to be a punctuation character. It is considered to be a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the space, comma, semicolon, or period separators.

The rules established for the formation of separators do not apply to the characters which comprise the contents of nonnumeric literals, comment entries, or comment lines.

Character-Strings

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

DBCS Support

You can use DBCS character-strings to form literals and comments. DBCS character-strings are constructed using characters from the Double-Byte Character Set of the system. DBCS character-strings can be embedded into nonnumeric strings.

End of DBCS Support

COBOL Words

A COBOL word is a character-string of not more than 30 characters that forms a user-defined word, system name, or reserved word. Within a given source program, these classes of COBOL words form disjoint sets. A COBOL word may belong to one and only one of these classes.

The same COBOL word may be used as a system name and as a user-defined word within the source program, provided the system name has not been set up as a reserved word. The class of a specific occurrence of this COBOL word is determined by the context of the clause or phrase in which it occurs.

VSC2

User-Defined Words

A user-defined word is a COBOL word supplied by the user to satisfy the format of a clause or statement. Each character of a user-defined word is selected from the set of characters A, B, C, ... Z, a, b, c, ... z (interpreted as equivalent to uppercase), numbers 0, ... 9, and -. The - may not appear as the first or last character.

Implemented user-defined word types are as follows:

alphabet name	
condition name	
<i>constant name</i>	<i>MF</i>
data-name	
file name	
index name	
level number	
library name	
mnemonic name	
paragraph name	
program name	
record name	
routine name	
section name	
segment number	
<i>split-key name</i>	<i>MF</i>
text name	

Within a given source program, 15 of these 17 types of user-defined words are grouped into 12 disjoint sets. The following are disjoint sets:

alphabet names	
cd names	
condition names, <i>constant names</i> , data-names,	<i>MF</i>
record names, <i>split-key names</i>	<i>MF</i>
file names	
index names	
library names	
mnemonic names	
paragraph names	
program names	
routine names	
section names	
text names	

All user-defined words, except segment numbers and level numbers, can belong to only one of these disjoint sets. Further, all user-defined words within a disjoint set must be unique. Refer to "Uniqueness of Reference" on page 2-28.

With the exception of paragraph name, section name, level number and segment number, all user-defined words must contain at least one alphabetic character *or one occurrence of the hyphen character*. *MF*

Segment numbers and level numbers need not be unique. Specification of a segment number or level number may be identical to any other segment number or level number and may even be identical to a paragraph name or section name.

Condition Name

A condition name is a name which is assigned to a specific value, set of values, or range of values, within a complete set of values that a data item may assume. The data item itself is called a conditional variable.

Condition names may be defined in the Data Division *or* in the SPECIAL-NAMES paragraph within the Environment Division, where a condition name must be assigned to the ON STATUS or OFF STATUS, or both, of the run-time switches.

A condition name is used only in the RERUN clause or in conditions as an abbreviation for the relation condition. This relation condition states that the associated conditional variable is equal to one of the set of values to which the condition name is assigned.

Constant Name

A constant name is a name which is assigned as the name of a fixed value. *MF*

Mnemonic Name

A mnemonic name assigns a user-defined word to an implementer name. These associations are established in the SPECIAL-NAMES paragraph of the Environment Division. Refer to "SPECIAL-NAMES Paragraph" on page 5-8.

Paragraph Name

A paragraph name is a word that names a paragraph in the Procedure Division.

Section Name

A section name is a word that names a section in the Procedure Division.

Other User-Defined Words

Refer to the Glossary for definitions of all other types of user-defined words.

System Names

A system name is a COBOL word used to communicate with the operating environment. Each character used in the formation of a system name must be selected from the set of characters A, B, C, ... Z, a, b, ... z, 0, ... 9 and -, except that the - may not appear as the first or last character.

There are three types of system names:

- computer name
- implementer name
- language name.

Within a given implementation, these three types of system names form disjoint sets. A given system name may belong to one and only one of them.

Refer to the Glossary for the individually defined system names.

Reserved Words

A reserved word is a COBOL word on a specified list that may be used in COBOL source programs, but must not appear in the programs as a user-defined word or system name. Reserved words can only be used as specified in the general formats.

There are six types of reserved words:

- Key words
- Optional words
- Connectives
- Special registers
- Figurative constants
- Special character words.

Key Words

A key word is a word required when the format in which the word appears is used in a source program. Within each format, key words are uppercase and underlined.

Key words are of three types:

- Verbs such as ADD, READ, and ENTER
- Required words that appear in statement and entry formats

-
- Words that have a specific functional meaning, such as **NEGATIVE**, **SECTION**, and so on.

Optional Words

Within each format, uppercase words not underlined are called optional words and may appear at your discretion. The presence or absence of an optional word does not alter the semantics of the COBOL program in which it appears.

Connectives

There are three types of connectives:

- Qualifier connectives used to associate a data-name, condition name, and text name, or a paragraph name with its qualifier: **OF**, **IN**.
- Series connectives that link two or more consecutive operands: **,** (separator comma) or **;** (separator semicolon).
- Logical connectives used in the formation of conditions: **AND**, **OR**.

Special Registers

Certain words are used to name and reference special registers. Special registers are memory areas created by the AIX VS COBOL system. The primary use of special registers is to store information produced in conjunction with the use of specific COBOL features. These special registers include **LINAGE COUNTER** (refer to Chapter 8, “File Input and Output”) and **DEBUG-ITEM** (refer to Chapter 17, “Program Debugging”).

Figurative Constants

Figurative constants are used to name and reference specific constant values. These reserved words are specified in Appendix D, “Reserved Word List.”

Special Character Words

The arithmetic operators and relation characters are reserved words. Refer to the Glossary.

Literals

A literal is one of the following:

- A character-string whose value is implied by the ordered set of characters of which it is composed, or
- A reserved word that references a figurative constant, or
- *A user-defined word that references a constant value.* *MF*

Every literal belongs to one of two types: nonnumeric or numeric.

DBCS Support

There is a DBCS type of literal.

End of DBCS Support

Nonnumeric Literals

A nonnumeric literal is a character-string delimited at both ends by quotation marks *or apostrophes* and consisting of any allowable character in the character set. Nonnumeric literals may be of 1 to 160 characters in length. Whether quotation marks or apostrophes are used as delimiters, the presence of that delimiter within a nonnumeric literal may be represented by two contiguous occurrences. The presence of the character not serving as the delimiter is represented by a single occurrence. The value of a nonnumeric literal in the object program is the string of characters itself, except:

OSVS VSC2

- The delimiting quotation marks are excluded.
- Each embedded pair of contiguous delimiter characters represents a single character.

All other punctuation characters are part of the value of the nonnumeric literal rather than separators. All nonnumeric literals are category alphanumeric. Refer to "PICTURE Clause" on page 6-18.

In addition, hexadecimal binary values can be attributed to nonnumeric literals by expressing literals as:

MF

X'nn' or x'nn'

where n is a hexadecimal character in the set 0-9 A-F a-f. nn may be repeated up to 120 times, but the number of hexadecimal digits must be even. The X may be uppercase or lowercase.

DBCS Support

In nonnumeric literals, Single-Byte Character Set (SBCS) and DBCS characters can be mixed within a character-string. AIX VS COBOL statements process mixed strings without sensitivity to the machine representation. Those statements that operate on a byte-to-byte basis (for example, STRING and UNSTRING) may result in strings that are not valid mixtures of SBCS and DBCS. It is the user's responsibility to be certain that the statements are used correctly.

Nonnumeric literals are specified as PIC X items and are not protected from mid-DBCS-character splitting. A literal declared as PIC G is a DBCS literal, and must contain only DBCS characters. The DBCS (PIC G) literals are protected from mid-DBCS-character splitting.

End of DBCS Support

Numeric Literals

Numeric literals are character-strings whose characters are selected from the digits 0 through 9, the plus sign, the minus sign, and/or the decimal point. The implementation allows for numeric literals of 1 to 18 digits in length. The rules for the formation of numeric literals are as follows:

1. A literal must contain at least one digit.
2. A literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.
3. A literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer.

If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a nonnumeric literal and it is treated as such by the AIX VS COBOL system.

-
4. The value of a numeric literal is the algebraic quality represented by the characters in the numeric literal. Every numeric literal is category numeric. Refer to "PICTURE Clause" on page 6-18.

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

In addition, hexadecimal binary values can be attributed to numeric literals by expressing literals as: *MF*

H'nn' or h'nn'

where n is a hexadecimal character in the set 0-9 A-F a-f. nn may be repeated up to 8 times, but the number of hexadecimal digits must be even. The H may be uppercase or lowercase.

DBCS Support

DBCS Literals

DBCS literals have the following format:

▶— G"DBCS-string" —▶

G Is the literal type designator for a DBCS literal

" Is the opening and closing delimiter

DBCS-string

Represents DBCS characters.

In general, the rules for forming a nonnumeric literal also apply to DBCS literals. The maximum length of DBCS literals, however, is 28 double-byte characters, and they may not be continued across lines.

End of DBCS Support

Figurative Constant Values

Figurative constant values are generated by the AIX VS COBOL system and referred to using the reserved words given below. These words must not be bounded by quotation marks when used as figurative constants. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

The figurative constant values and the reserved words used to reference them are shown in Table 2-1.

Table 2-1. Figurative Constant Values and the Reserved Words	
CONSTANT	REPRESENTATION
ZERO ZEROS ZEROES	Represents the value 0, or one or more of the character 0 depending on context.
SPACE SPACES	Represents one or more of the character space from the computer's character set.
HIGH-VALUE HIGH-VALUES	Represents one or more of the character that has the highest ordinal position in the program collating sequence. (Hex FF for the ASCII character set.)
LOW-VALUE LOW-VALUES	Represents one or more of the character that has the lowest ordinal position in the program collating sequence. (Hex 00 for the ASCII character set.)
QUOTE QUOTES	Represents one or more of the character ". The word QUOTE or QUOTES cannot be used in place of a quotation mark in a source program to bound a nonnumeric literal. Thus, QUOTEABD QUOTE is incorrect as a way of stating the nonnumeric literal "ABD".
ALL literal	Represents one or more characters of the string of characters comprising the literal. The literal must be either a nonnumeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only.
NULL NULLS	<i>Represents one or more unset pointer values. A pointer variable with the NULL value is guaranteed not to point to any data item.</i> VSC2

DBCS Support

The figurative constant values for DBCS programs are:

HIGH-VALUE, HIGH-VALUES	hex FFFF
ZERO, ZEROS, ZEROES	hex 824F
LOW-VALUE, LOW-VALUES	hex 0000
QUOTE, QUOTES	hex 818D
SPACE, SPACES	hex 2020

End of DBCS Support

When a figurative constant represents a string of one or more characters, the length of the string is determined by the AIX VS COBOL system from context according to the following rules:

1. When a figurative constant is associated with another data item, as when the figurative constant is moved to or compared with another data item, the string of characters specified by the figurative constant is repeated character by character on the right until the size of the resulting string is equal to the size, in characters, of the associated data item. This is done prior to, and independent of, the application of any JUSTIFIED clause associated with the data item.
2. When a figurative constant is not associated with another data item, as when the figurative constant appears in a DISPLAY, STRING, STOP, or UNSTRING statement, the length of the string is one character.

DISPLAY SPACE in Format 2 of the DISPLAY statement is an exception.

MF

A figurative constant may be used wherever a literal appears in a format, except that whenever the literal is restricted to having only numeric characters in it, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

When the figurative constants HIGH-VALUES(S) or LOW-VALUE(S) are used in the source program, the actual character associated with each figurative constant depends upon the program collating sequence specified. Refer to "OBJECT-COMPUTER Paragraph" on page 5-6 and "SPECIAL-NAMES Paragraph" on page 5-8.

Constant Names

Constant names are user-defined words described in the Data Division in level-78 data description entries. A constant name may be used wherever a literal appears in a format. The effect of a constant name is the same as the literal assigned in its data description. A constant name with an integer value can also be used wherever a format requires an integer. For example, a constant may be used in place of a level number or segment number, or in a PICTURE character-string.

MF

A constant name can only be used after it has been described. It cannot be the object of a forward reference.

Special Registers

Special registers are data items generated by the AIX VS COBOL system and referred to by using their associated names (see Table 2-2). These special registers are subject to special rules of reference and have implicit data descriptions (PICTURES), as individually described.

OSVS VSC2

Special Register	Implicit Data Description Picture	Usage
CURRENT-DATE	X(8)	The CURRENT-DATE special register contains the value of the current date (as supplied by the COBOL program execution environment) in the form: MMDDYY where MM is the month number, DD is the day of the month, and YY is the year number (from 1900). CURRENT-DATE is valid only as the sending area of a MOVE statement.
TALLY	9(5) COMP	The TALLY special register contains information produced by the EXAMINE ... TALLYING statement. It is valid as a data-name in a Procedure Division statement wherever an elementary data item may be referenced.

OSVS VSC2

Table 2-2 (Page 2 of 3). Special Registers, Implicit Data Description Picture, and Usage		
Special Register	Implicit Data Description Picture	Usage
<i>TIME-OF-DAY</i>	<i>9(6) DISPLAY</i>	<p>The <i>TIME-OF-DAY</i> special register contains the value of the current time of day (24-hour clock) (as supplied by the COBOL program execution environment) in the form: hhmmss where hh = hour, mm = minutes, and ss = seconds. <i>TIME-OF-DAY</i> is valid only as the sending area of a <i>MOVE</i> statement.</p> <p style="text-align: right;">OSVS</p>
<i>RETURN-CODE</i>	<i>S9(4) COMP (This can be changed by the RTNCODE-SIZE directive.) MF</i>	<p>The <i>RETURN-CODE</i> special register may:</p> <ul style="list-style-type: none"> • Be set by a program, prior to the execution of a <i>STOP RUN</i>, <i>EXIT PROGRAM</i> or <i>GOBACK</i> statement, to pass a value to the invoking program (or the execution environment). • Be read, subsequent to a <i>CALL</i> to another COBOL program, to obtain the <i>RETURN-CODE</i> set by that <i>CALLed</i> program. <p>A program's <i>RETURN-CODE</i> is set to 0 when the program is first entered. The <i>RETURN-CODE</i> is valid as a data-name in a Procedure Division statement wherever an elementary data item may be referenced. If a program using a 2-byte <i>RETURN-CODE</i> returns to a program using a 4-byte <i>RETURN-CODE</i>, the top 2 bytes of the calling program's <i>RETURN-CODE</i> will be undefined.</p> <p style="text-align: right;">OSVS VSC2</p>
<i>WHEN-COMPILED</i>	<i>X(20)</i>	<p>The <i>WHEN-COMPILED</i> special register contains the time and date that the COBOL program was submitted to the AIX VS COBOL system, in the form: hh.mm.ssMMM DD, YYYY where hh = hours (24-hour clock), mm = minutes, ss = seconds, MMM = month name (first 3 characters), DD = day of month, and YYYY = year.</p> <p><i>WHEN-COMPILED</i> is valid only as the sending area of a <i>MOVE</i> statement.</p> <p style="text-align: right;">OSVS</p>

Table 2-2 (Page 3 of 3). Special Registers, Implicit Data Description Picture, and Usage		
Special Register	Implicit Data Description Picture	Usage
<i>WHEN-COMPILED</i>	X(20)	<i>The WHEN-COMPILED special register contains the time and date that the COBOL program was submitted to the AIX VS COBOL system in the form: MM/DD/YYhh.mm.ss where DD, hh, mm and ss are as above. YY = year in century and MM = month in year.</i> <i>WHEN-COMPILED is valid only as the sending area of a MOVE statement.</i>
<i>SORT-MESSAGE</i> <i>SORT-FILE-SIZE</i> <i>SORT-MODE-SIZE</i> <i>SORT-CORE-SIZE</i> <i>SORT-CONTROL</i>	X(8) S9(8) COMP S9(5) COMP S9(8) COMP X(8)	<i>These items may be referenced in the Procedure Division but will contain zero (except SORT-MESSAGE and SORT-CONTROL which will contain spaces).</i>
<i>SORT-RETURN</i>	S9(4) COMP	<i>SORT-RETURN may be used to cause an abnormal termination of a SORT procedure. If a value of 16 is moved into this field, the SORT operation will be terminated after the next RELEASE or RETURN.</i>
<i>ADDRESS</i>	USAGE IS POINTER	<i>An ADDRESS special register exists for each 01 and 77 level item in the LINKAGE and WORKING-STORAGE SECTION. The value of the special register is the address of the record.</i>

Each reserved word used to refer to a figurative constant value is a distinct character-string with the exception of the construction ALL literal, which is composed of two distinct character-strings.

PICTURE Character-Strings

A PICTURE character-string consists of certain combinations of characters in the COBOL character set used as symbols. Refer to "PICTURE Clause" on page 6-18 for the PICTURE character-string and the rules that govern its use.

Any punctuation character that appears as part of the specification of a PICTURE character-string is not considered a punctuation character, but rather a symbol used in the specification of that PICTURE character-string.

Comment-Entries

A comment-entry in the Identification Division may be any combination of characters from the character set of the computer.

DBCS Support

Character-strings that form comments may contain either DBCS characters or a combination of DBCS and SBCS characters. Multiple comment lines containing DBCS strings are allowed. The embedding of DBCS characters in a comment line must be done on a line-by-line basis. DBCS characters cannot be continued to a following line.

End of DBCS Support

Concept Of Computer-Independent Data Description

To make data as computer independent as possible, the characteristics or properties of the data are described in relation to a standard data format rather than an equipment oriented format. This standard data format is oriented to general data processing applications, and uses the decimal system to represent numbers (regardless of the radix used by the computer) and the remaining characters in the AIX VS COBOL character set to describe non-numeric data items.

Concept Of Levels

A level concept, or hierarchy, is inherent in the structure of a logical data record. This concept arises from the need to specify subdivisions of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referral.

The most basic subdivisions of a record, those not further subdivided, are called elementary items. Consequently, a record is said to consist of a sequence of elementary items, or the record itself may be an elementary item.

In order to refer to a set of elementary items, the elementary items are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into groups of two or more groups, etc. Thus, an elementary item may belong to more than one group.

Level Numbers

A system of level numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level numbers for records start at 01. Less inclusive data items are assigned higher (not necessarily successive) level numbers not greater in value than 49. A maximum of 49 levels in a record is allowed. There are special level numbers (66, 77, 78, and 88) which are exceptions to this rule (see below). Separate entries are written in the source program for each level number used.

MF

A group includes all group and elementary items following it until a level number less than or equal to the level number of that group is encountered. All items which are immediately subordinate to a given group item should be described using identical level numbers greater than the level number used to describe that group item. *This rule is not insisted upon.*

OSVS VSC2

Correct	Incorrect but Permitted	
01 A.	01 A.	
05 C-1.	05 C-1.	
10 D PICTURE X.	10 D PICTURE X.	
10 E PICTURE X.	10 E PICTURE X.	
05 C-2.	04 B-1.	OSVS VSC2

Figure 2-1. Example of Level Numbers in Group Descriptions

Three types of entries exist for which there is no true concept of level. These are:

- Entries that specify elementary items or groups introduced by a RENAME clause
- Entries that specify noncontiguous working storage and linkage data items
- Entries that specify condition names.

Entries describing items by means of RENAME clauses for the purpose of regrouping data items have been assigned the special level number 66.

Entries that specify noncontiguous data items, which are not subdivisions of other items, and are not themselves subdivided, have been assigned the special level number 77.

Entries that specify constant names, to be associated with the value of a particular literal, have been assigned the special level number 78. MF

Entries that specify condition names, to be associated with particular values of a conditional variable, have been assigned the special level number 88.

For an example of level numbers representing a data hierarchy, see Figure 2-2.

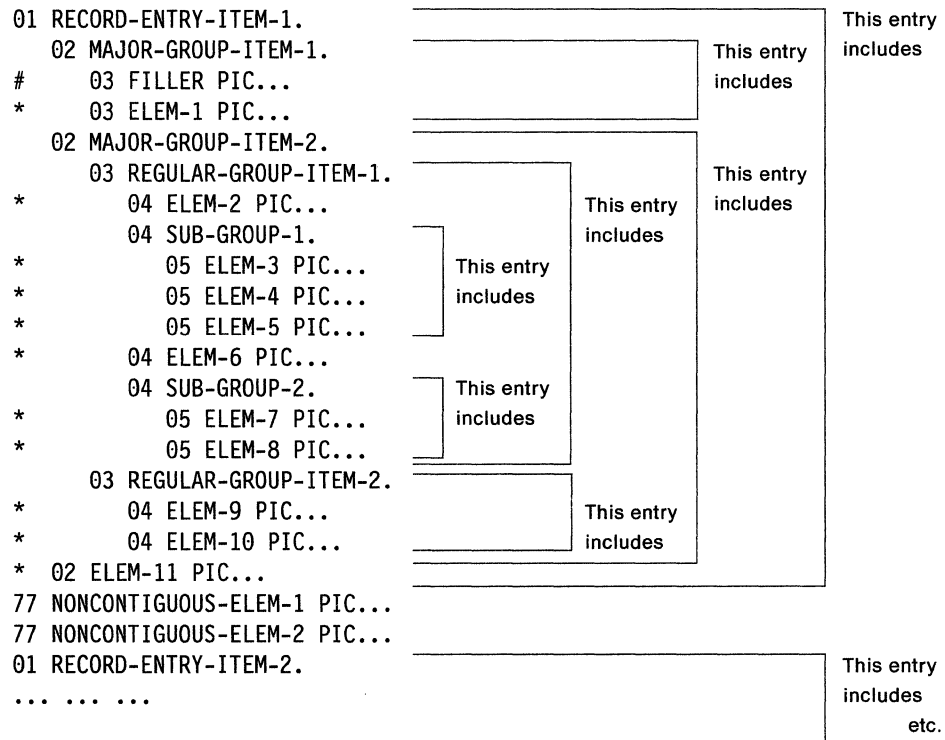


Figure 2-2. Example of Level Numbers Representing a Data Hierarchy

Note that indentation of COBOL source code is only a readability convention and is not part of the language.

Elementary items are by definition those items with no subordinate entries (entries with no numerically greater level numbers) following, and must have a storage definition associ-

ated with them. Refer to "PICTURE Clause" on page 6-18 and "USAGE Clause" on page 6-39.

Only elementary items (marked with an asterisk, *, above) and FILLER items (marked with a # sign above) will have storage explicitly reserved for them (in accordance with "PICTURE Clause" on page 6-18). Nonelementary items have implicit storage associated with them of size determined by their subordinate items plus any FILLER bytes needed for synchronization. Refer to "SYNCHRONIZED Clause" on page 6-37.

Level numbers need not be consecutively ascending or descending as shown in Figure 2-2 on page 2-17 for clarity. Thus, the next subordinate level after 01 could be 05, and the next level 10, and so on.

The data record in Figure 2-2 on page 2-17 would produce storage allocation in the following manner:

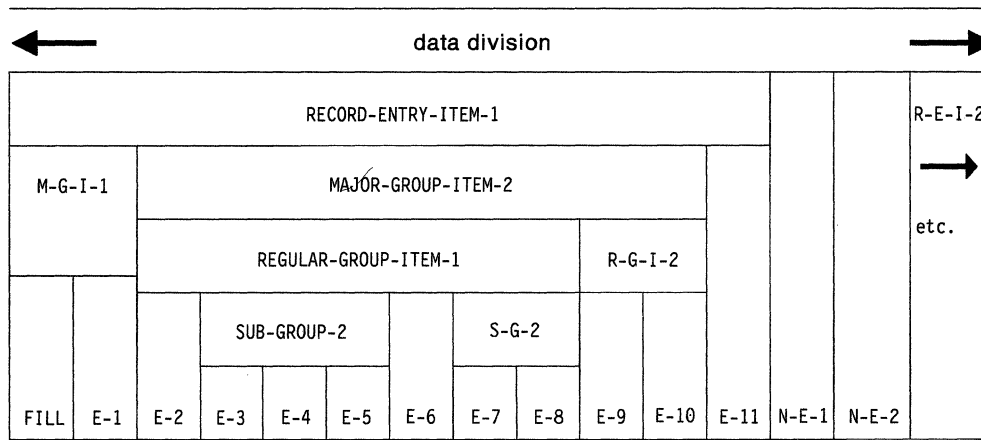


Figure 2-3. Storage Allocation

Concepts of Classes of Data

The five categories of data items (refer to "PICTURE Clause" on page 6-18) are grouped into four classes: alphabetic, numeric, DBCS, and alphanumeric. For alphabetic, DBCS, and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric-edited, numeric-edited, and alphanumeric (without editing). Every elementary item, except for an index data item, belongs to one of the classes and to one of the categories. The class of a group item is treated at object time as alphanumeric regardless of the class of elementary items subordinate to the group item. The relationship of the class and categories of data items are depicted in Table 2-3 on page 2-19.

Table 2-3. Data Levels, Classes, and Categories		
Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric-Edited Alphanumeric-Edited Alphanumeric
	DBCS	DBCS
Nonelementary Group	Alphanumeric	Alphabetic Numeric Numeric-Edited Alphanumeric-Edited Alphanumeric DBCS

Algebraic Signs

Algebraic signs fall into two categories: operational signs, for indicating algebraic properties of signed numeric data items and signed numeric literals; and editing signs, for identifying the sign of the item in edited reports.

The SIGN clause permits the programmer to explicitly state the location of the operational sign. The clause is optional. If it is not used, operational signs are defined by setting bit 6 of the trailing digit for ASCII numbers (see Table 2-4).

Editing signs are inserted into a data item by using the sign control symbols of the PICTURE clause.

Standard Alignment Rules

The standard rules for positioning data within an elementary item depend on the category of the receiving item. These rules are:

1. If the receiving data item is described as numeric:
 - a. The data is aligned by decimal point and is moved to the receiving character positions with zero-fill or truncation on either end as required.
 - b. When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its rightmost character and is aligned as in item 1a above.
2. If the receiving data item is a numeric-edited data item, the data moved to the edited item is aligned by decimal point with zero-fill or truncation at either end, as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.
3. If the receiving data item is alphanumeric (other than a numeric-edited data item), alphanumeric-edited or alphabetic, the sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item with space fill or truncation to the right, as required.

DBCS Support

4. For DBCS receiving items, the data is aligned at the leftmost character position, and (if necessary) truncated or padded with DBCS spaces at the right.

End of DBCS Support

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified as described in “JUSTIFIED Clause” on page 6-13.

Item Alignment for Increased Object Code Efficiency

Some computer memories are organized to have natural addressing boundaries in the computer memory (for example, word boundaries, half-word boundaries, byte boundaries). The object program determines the way in which data is stored and need not respect these natural boundaries.

However, certain uses of data (for example, in arithmetic operations or in subscripting) may be facilitated if the data is stored so as to be aligned on these boundaries. Specifically, additional machine operations for accessing and storing data may be repeated if portions of two or more data items appear between adjacent natural boundaries, or if certain natural boundaries divide a single data item.

Data items aligned on these natural boundaries to avoid additional machine operations are defined as synchronized. A synchronized item is assumed to be introduced and carried in that form. However, conversion to synchronized form occurs only during the execution of a statement other than READ or WRITE which stores data in the item.

Synchronization can be accomplished in two ways:

- By use of the SYNCHRONIZED clause.
- By organizing the data suitably on the appropriate natural boundaries without using the SYNCHRONIZED clause.

By using the SYNCHRONIZED clause, special types of alignment within a group may affect the results of statements in which the group is used as an operand. The effect of the implicit FILLER and the semantics of any statement referencing these groups are described in “Implicit FILLER or Padding Bytes” on page 2-22 and “Example of Implicit FILLER Assignments” on page 2-23.

Selection of Character Representation and Radix

The value of a numeric item, which is defined as numeric by its PICTURE (refer to “PICTURE Clause” on page 6-18), may be represented in memory in either binary or decimal form depending on the USAGE clause of the declaration. Refer to “USAGE Clause” on page 6-39. These numeric formats are:

- DISPLAY
- COMPUTATIONAL, COMP, BINARY
COMPUTATIONAL-4 or COMP-4 VSC2 OSVS
- *COMPUTATIONAL-3, COMP-3 or PACKED-DECIMAL* MF OSVS
VSC2
- *COMPUTATIONAL-5 or COMP-5* MF
- POINTER VSC2
- *COMPUTATIONAL-X or COMP-X* MF

DISPLAY Format

The numeric digit characters that represent the number value for the display format are held in radix 10, one digit character per byte of computer memory, in ASCII representation. This is the standard data format of the COBOL language. If the data item is signed and the sign is not specified as SEPARATE (refer to "SIGN Clause" on page 6-35) the numeric sign is incorporated into either the leading or trailing digit, according to the LEADING or TRAILING phrase in the SIGN clause. Sign data is incorporated into the requisite digit as shown in Table 2-4 (bit 6 (value 40) of the character is set from zero to one if the whole number has a negative value). If the data item is signed and the sign is specified as SEPARATE, then the sign is held as a separate single character, additional to the digits, either ASCII character + or - as necessary. This sign character appears as the leading or trailing byte of the stored numeric data item according to the LEADING or TRAILING phrase of the sign clause.

Leading or Trailing Value Digit Before Sign Incorporation	Sign Digit Character (and hexadecimal value) for:	
	Positively-Signed Values	Negatively-Signed Values
0	0(30)	p(70)
1	1(31)	q(71)
2	2(32)	r(72)
3	3(33)	s(73)
4	4(34)	t(74)
5	5(35)	u(75)
6	6(36)	v(76)
7	7(37)	w(77)
8	8(38)	x(78)
9	9(39)	y(79)

Storage character position requirements for DISPLAY data items are equal to the number of 9s in the PICTURE clause plus one if the sign is specified as SEPARATE. The SYNCHRONIZED clause has no effect on DISPLAY format data declarations.

COMPUTATIONAL, COMP, BINARY, COMPUTATIONAL-4, or COMP-4 Format

These numeric data items are held in computer memory in pure binary two's complement representation. In this format, number values are held in radix of two where each computer bit in the representation starting from the right (least significant) and represents the presence or absence of an increasingly significant power of two in that value. Negative numbers are represented by complementing (inverting all the bit values of) their positive counterpart, and then adding one to the whole. Storage requirements depend on the number of 9s in the PICTURE clause and whether the numeric data item is signed or unsigned (refer to "PICTURE Clause" on page 6-18, and "SIGN Clause" on page 6-35). The AIX VS COBOL system assigns storage for COMPUTATIONAL items in one of two modes: byte storage and word storage. Byte storage is the default storage assignment mode for the AIX VS COBOL language.

Computer Memory Natural Boundaries

The fundamental natural boundaries of computer memory are usually based on an 8-bit character known as a byte. Within this fundamental framework, machines fall into two broad categories: those with no other natural boundaries (byte storage computers) and those with other natural boundaries based on multiples of the fundamental boundary of the byte (word storage computers).

In byte storage mode, AIX VS COBOL assigns numeric storage such that each numeric item occupies the minimum number of bytes (refer to “Selection of Character Representation and Radix” on page 2-20). The SYNCHRONIZED clause has no meaning in the context and hence has no effect. Byte storage is the default storage assignment mode for AIX VS COBOL.

Within word storage computers natural boundaries may occur at 2-byte, 4-byte and/or 8-byte boundaries. Accordingly, the VS COBOL language can provide data item storage assignment and synchronization when the COMPUTATIONAL clause and possibly the SYNCHRONIZED clause are used. This word storage assignment of COMPUTATIONAL format data is controlled by an AIX VS COBOL system directive, IBMCOMP. Refer to the *User's Guide* for information on how to invoke this feature.

Number of Digits (9s) in PICTURE Representation		Character-Positions (Bytes) of Storage Assigned	
Signed	Unsigned	Byte Storage Mode	Word Storage
1-2	1-2	1	2
3-4	3-4	2	2
5-6	5-7	3	4
7-9	8-9	4	4
10-11	10-12	5	8
12-14	13-14	6	8
15-16	15-16	7	8
17-18	17-18	8	8

Note: Byte storage is the default storage assignment mode for VS COBOL. For details of how to enable the word storage feature of the AIX VS COBOL system using the IBMCOMP system directive, refer to the *User's Guide*.

Implicit FILLER or Padding Bytes

When the word storage assignment mode is enabled, and a numeric data item is specified as COMPUTATIONAL, extra character positions or bytes of computer memory may be assigned to that data item (refer to Figure 2-4 on page 2-23). These bytes are known as padding, or implicit FILLER bytes, and are not normally accessible to the program. Similar implicit FILLER bytes can be generated by use of the SYNCHRONIZED clause.

Synchronization

If a data item description contains the SYNCHRONIZED clause, and word storage mode is enabled, then the position of that item within the computer memory is aligned such that the right (least significant) end is on a natural boundary of the memory. Extra character positions (bytes) of computer memory are reserved adjacent to synchronized items to achieve this alignment. These bytes, known as padding bytes or implicit FILLER bytes, are normally inaccessible to the computer program.

Each elementary data item described as SYNCHRONIZED is aligned to the natural memory boundary corresponding to its data item storage assignment (according to Table 2-5). Thus, in word storage mode, a numeric data item with a PICTURE description

of S9 (5) is assigned 4 bytes of memory (1 padding byte and 3 data bytes). If SYNCHRONIZED was specified, it is aligned to the next nearest 4-byte boundary. The total (4-byte) memory assignment is aligned so the number of bytes from the beginning of the record containing the item to the left (most significant) end of the item was a multiple of four. If the previous item does not end on a 4-byte boundary, then implicit FILLER assignments are necessary to achieve this.

Other such implicit FILLER bytes may be generated by using the SYNCHRONIZED items in nonelementary data descriptions containing an OCCURS clause. Refer to "OCCURS Clause" on page 12-5. This is because extra bytes may need to be reserved for each group item occurrence so that the second or subsequent occurrences have the same alignment to the natural boundaries of the computer memory as did the first occurrence.

Implicit Synchronization

With word storage mode enabled, all record level data descriptions are automatically synchronized to a full 8-byte boundary.

Example of Implicit FILLER Assignments

The following COBOL data description produces the computer memory allocation shown in Figure 2-4. An explanation of the symbols used in the figure is shown below it.

```

01 UNSYNCHRONIZED-RECORD.
   02 UNSYNCHRONIZED-DATA-1 PIC 9(3) DISPLAY.
   02 UNSYNCHRONIZED-DATA-2 PIC X(2).
01 COMPOUND-REPEATED-RECORD.
   02 ELEMENTARY-ITEM-1 PIC X(2).
   02 GROUP-ITEM OCCURS 3 TIMES.
     03 ELEMENTARY-ITEM-2 PIC X.
     03 ELEMENTARY-ITEM-3 PIC S9(2) COMP SYNC.
     03 ELEMENTARY-ITEM-4 PIC S9(4) V9(2) COMP SYNC.
     03 ELEMENTARY-ITEM-5 PIC X (5).

```

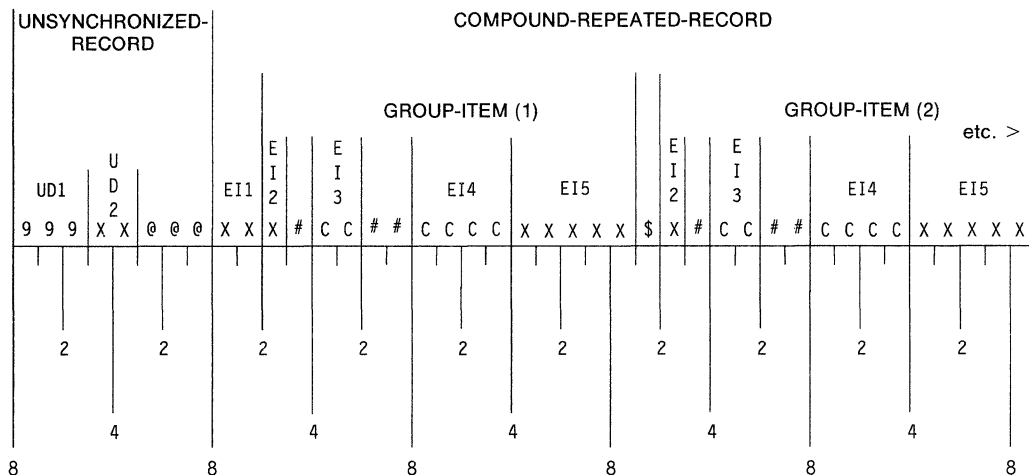


Figure 2-4. Sample Computer Storage Allocation

where:

- @ indicates implicit FILLER bytes allocated due to automatic synchronization of a record (01-level) description.
- # indicates implicit FILLER bytes allocated when following data item is explicitly synchronized.
- \$ indicates implicit FILLER bytes allocated when a nonelementary item is subject to an OCCURS clause.

- 9 indicates bytes allocated for a numeric DISPLAY character.
- X indicates bytes allocated for an alphanumeric DISPLAY character.
- C indicates bytes allocated for a COMPUTATIONAL data storage.

Truncation

In data items of USAGE COMP, data is held in binary format as described in the previous sections. The memory allocated for an item may have space for larger numbers than specified by the PICTURE clause. For example, an item described as PIC 99 COMP is normally assigned one byte, which can hold numbers up to 255.

To conform with the rules of ANSI COBOL, numbers behave as decimal numbers, regardless of their format. In an arithmetic statement, if the result is bigger than the PICTURE clause of a receiving item allows, this causes a size error, and if the ON SIZE ERROR phrase is specified the result is not stored in the receiving item. In a nonarithmetic statement, if this situation occurs, the decimal value is truncated on the left to the number of digits specified in the PICTURE clause.

However, data in USAGE COMP items can be forced to behave as binary data. Truncation only occurs if it is necessary in order for the data to fit the space allocated. The behavior of USAGE COMP items is controlled by the setting of an AIX VS COBOL system directive, TRUNC. Refer to the User's Guide for details on how to invoke this feature. This directive selects whether the decimal value is truncated to the picture size, or the binary value is truncated to the space available. It distinguishes between results of arithmetic statements and data being moved by nonarithmetic statements. MF

Regardless of the setting of the directive, an arithmetic statement gives the size error condition if the result has more decimal digits than specified in the PICTURE clause of a receiving item. MF

Example of Truncation

The TRUNC directive can change the results of some operations, as demonstrated in the following examples in which item A is described as PIC 99 COMP. MF

Operation	Result		
	TRUNC	NOTRUNC	TRUNC "ANSI"
MOVE 163 TO A	63	163	63
MOVE 263 TO A	63	7	63
MOVE 13 TO A, ADD 150 TO A	63	163	undefined results
MOVE 13 TO A, ADD 250 TO A	63	7	undefined results

Notes:

MF

1. *This directive has no effect on the truncation of low order digits in non-integer data. This always conforms with the behavior specified in ANSI COBOL.*
2. *If the IBMCOMP system directive is set, extra upper bytes may be allocated to a COMP item. These are counted in the space allocated. When IBMCOMP is on, padding bytes may be generated before a COMP item with a SYNC clause. These are not part of the item, and are never affected by data stored in the item.*
3. *When a value being stored into a signed item is limited to the number of digits allowed by the PICTURE clause, it can never be big enough to overwrite the sign bit. When NOTRUNC is set, the value, if large enough, will overwrite the sign bit.*

COMPUTATIONAL-3 COMP-3 PACKED-DECIMAL Format

This form, commonly called binary-coded-decimal form, represents numeric data items in radix 10, but with each digit of the value held in only one-half of one computer character, as described in Table 2-7. The sign is held in a separate trailing digit (half-character) position; that is, at the right (least significant) end of the item.

Digit Value	Digit Representation in Hexadecimal	
	Left Half-Character (odd-digit)	Right Half-Character (even-digit)
0	X'00'	X'00'
1	X'10'	X'01'
2	X'20'	X'02'
3	X'30'	X'03'
4	X'40'	X'04'
5	X'50'	X'05'
6	X'60'	X'06'
7	X'70'	X'07'
8	X'80'	X'08'
9	X'90'	X'09'

Note: Count even and odd starting from the right.

Table 2-8 on page 2-26 shows the sign digits used for COMPUTATIONAL-3; storage memory requirements for this format depend only on the number of 9s in the PICTURE clause of the data item (see Table 2-9 on page 2-26).

Table 2-8. COMPUTATIONAL-3 Sign Digit Representation		
Sign Convention in the PICTURE Clause	Sign of Data Item Value	Sign Half-Character, in Hexadecimal
Unsigned	n/a	X'0F'
Signed	+	X'0C'
Signed	-	X'0D'

Table 2-9. Numeric Data Storage for the COMP(UTATIONAL)-3 or PACKED-DECIMAL PICTURE Clause	
Bytes Required	Number of Digits (Signed or Unsigned)
1	1
2	2-3
3	4-5
4	6-7
5	8-9
6	10-11
7	12-13
8	14-15
9	16-17
10	18

Examples

1. For COMPUTATIONAL-3 and PICTURE 9999, the number + 1234 would be stored as follows:

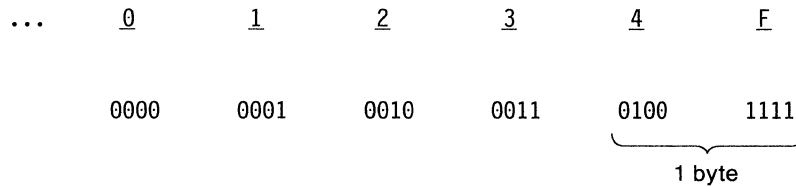


Figure 2-5. Number Storage

where F represents the nonprinting plus sign.

2. For COMPUTATIONAL-3 and PICTURE S9999, the number + 1234 is stored as in number 1 above, except that the least significant digit would be replaced by C(1100) representing the plus sign.
3. For COMPUTATIONAL-3 and PICTURE S9999, the number – 1234 is stored as in number 1 above, except that the least significant byte would be replaced by D(1101) representing the minus sign.

The SYNCHRONIZED clause (with or without the LEFT or RIGHT phrase) has no effect on COMPUTATIONAL-3 data declarations.

COMPUTATIONAL-5 Or COMP-5 Format

This format is the same as COMPUTATIONAL format. Refer to “COMPUTATIONAL, COMP, BINARY, COMPUTATIONAL-4, or COMP-4 Format” on page 2-21.

MF

It has the following differences from the COMPUTATIONAL format:

- *The value that can be stored is not limited to the number of decimal digits indicated in the PICTURE clause, but to the largest binary number for which the allocated storage has space.*
- *The machine code of some processors stores the bytes of numeric fields in reverse order. That is, low-order bytes are stored at the lowest addresses and successively higher-order bytes at successively higher addresses. This is the mirror image of normal order where the high-order bytes are stored at the lowest addresses and successively lower-order bytes at successively higher addresses.*
 - *On processors where the machine code stores the bytes of numeric fields in reverse order, COMPUTATIONAL-5 items are stored in reverse order. For example, hexadecimal 12 34 56 78 9A is stored as 9A 78 56 34 12.*
 - *On processors where the machine code stores the bytes of numeric fields in normal order, COMPUTATIONAL-5 items are stored in normal order.*

COMPUTATIONAL-X Or COMP-X Format

This format is the same as COMPUTATIONAL format. See “COMPUTATIONAL, COMP, BINARY, COMPUTATIONAL-4, or COMP-4 Format” on page 2-21.

MF

It has the following differences from the COMPUTATIONAL format:

- *The PICTURE character string can consist of all Xs. If it does, the number of Xs gives the length of the item in bytes.*
- *Whether the PICTURE character string consists of Xs or 9s, the value that can be stored is limited to the largest binary number for which the allocated storage has space. The item is not affected by the TRUNC, COMP, SYNC, ALIGN, or IBMCOMP system directives.*
- *The use of COMP-X items in arithmetic statements is restricted to ADD, SUBTRACT, MULTIPLY and DIVIDE statements with two operands (each either a literal or a COMP-X item) and no ON SIZE ERROR phrase. Refer to “ON SIZE ERROR Phrase and NOT ON SIZE ERROR Phrase” on page 7-19, “ADD Statement” on page 7-24, “DIVIDE Statement” on page 7-34, “MULTIPLY Statement” on page 7-69, and “SUBTRACT Statement” on page 7-91. Such statements follow these rules of two’s complement binary arithmetic:*
 - *If the result is too big for the target item, high-order binary digits are truncated.*
 - *If the result is not an integer, only the integer part is stored.*
 - *If the result is less than zero, the two’s complement of the absolute value of the result is stored. This is subsequently interpreted as a positive (unsigned) integer.*
 - *If a negative literal is used in a MULTIPLY or DIVIDE statement, its sign is ignored and it is treated as positive.*
- *If a nonarithmetic statement attempts to store a negative value in a COMP-X item, the absolute value is stored.*

MF

POINTER Format

VS COBOL assigns four bytes of storage for POINTER format. The method of data storage is machine-dependent.

VSC2

Uniqueness of Reference

Uniqueness of Reference contains the following categories:

- Qualification
- Subscripting
- Indexing
- Reference Modification
- Identifier
- Condition-Name.

Qualification

Every user-specified name in a COBOL source program must be unique, either because no other name has the identical spelling and hyphenation, or because the name exists within a hierarchy of names such that references to the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers and the process that specifies uniqueness is called qualification. Enough qualification must be mentioned to make the name unique. However, it may not be necessary to mention all levels of the hierarchy.

Within the Data Division, all data-names used for qualification must be associated with a level indicator or a level number. Therefore, two identical data-names must not appear as entries subordinate to a group item unless they are capable of being made unique through qualification.

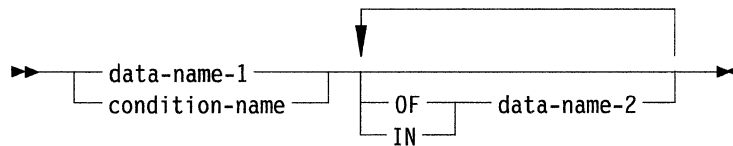
In the Procedure Division, two identical paragraph names must not appear in the same section.

In the hierarchy of qualification, names for level indicators are the most significant, followed by the names for level-number 01, followed by names for level-number 02 through 49. A section name is the only qualifier available for a paragraph name. Thus, the most significant name in the hierarchy must be unique and cannot be qualified. Subscripted or indexed data-names and conditional variables, as well as procedure names and data-names, may be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition names. Regardless of the available qualification, no name can be both a data-name and procedure name.

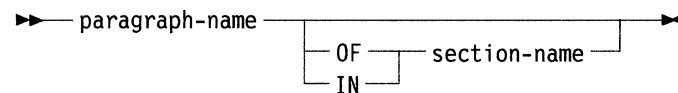
Qualification is performed by following a data-name, a condition name, a paragraph name, or a text name with one or more phrases composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent.

The following figure shows the general formats for qualification:

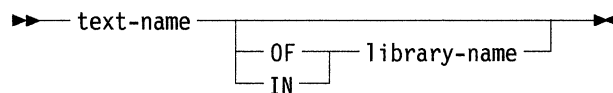
Format 1



Format 2



Format 3



The rules for qualification are as follows:

1. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.
2. The same name must not appear at two levels in a hierarchy.
3. If a data-name or a condition name is assigned to more than one data item in a source program, the data-name or condition name must be qualified each time it is referred to in the Procedure, Environment, and Data Divisions (except in the REDEFINES clause, where qualification is unnecessary and must not be used).
4. 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 from within the same section.
5. A data-name cannot be subscripted when it is being used as a qualifier.
6. A name can be qualified even though it does not need qualifications. If there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used.
Qualified names may have up to 50 qualifiers.
7. If more than one COBOL library is available to the AIX VS COBOL system, text name must be qualified each time it is referenced.

Subscripting

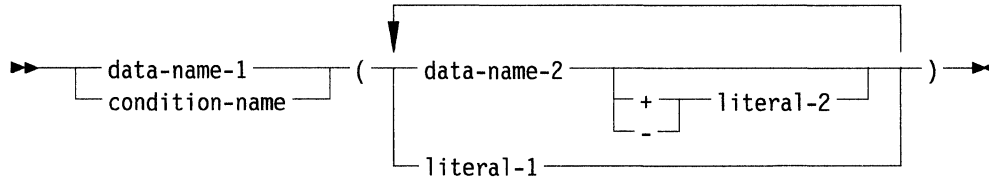
Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names. Refer to "OCCURS Clause" on page 12-5.

The subscript can be represented either by a numeric literal that is an integer, by a data-name, or by a data-name followed by the operator + or -, followed by an unsigned integer numeric literal. The data-name must be a numeric elementary item that represents an integer, and the whole subscript must be delimited by the balanced pair of separators left parenthesis and right parenthesis.

The subscript data-name may be signed. If it is signed, it must be positive. The lowest possible subscript value is 1. This value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, and so on. The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause.

The subscript, or set of subscripts, that identifies the table element is delimited by the balanced pair of separators, left parenthesis and right parenthesis, following the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier. When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization. Up to 16 subscripts are permitted.

The following figure shows the general format for subscripting:



where:

- literal-1 must be a positive integer literal.
- literal-2 must be an unsigned numeric integer.
- data-name-2 may not be subscripted or indexed.

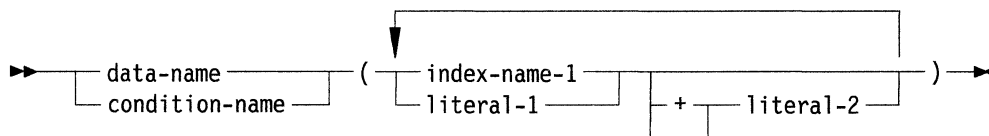
Indexing

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table using the INDEXED BY phrase in the definition of a table. A name specified in the INDEXED BY phrase is known as an index name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in any table. An index name must be initialized before it is used as a table reference. An index name can be given an initial value by a SET statement.

Direct indexing is specified using an index name in the form of a subscript. Relative indexing is specified when the index name is followed by the operators + or -, followed by an unsigned integer numeric literal, all delimited by the balanced pair of separators, left parenthesis and right parenthesis, following the table element data-name. The occurrence number resulting from relative indexing is determined by incrementing (where the operator + is used) or decrementing (where the operator - is used), by the literal value, or the occurrence number represented by the index value. When more than one index name is required, the names are written in the order of successively less inclusive dimensions of the data organization.

When a statement which refers to an indexed table element is executed, the value contained in the index referenced by the index name for the table element must neither correspond to a value less than one nor to a value greater than the highest permissible occurrence number of an element of the associated table. This restriction also applies to the value resulting from relative indexing. Up to 16 index names can be used with a data-name.

The following figure shows the general format for indexing:



where:

- literal-1 must be a positive numeric integer.
- literal-2 must be an unsigned numeric integer.

Reference Modification

Reference modification defines a data item by specifying a leftmost character and length for the data item. Unless otherwise specified, it is allowed anywhere an identifier referencing an alphanumeric data item is permitted. The general format for reference modification is:

data-name (leftmost-character-position: [length])

where:

Data-name may be qualified or subscripted and must reference a data item whose usage is DISPLAY.

leftmost-character-position and length must be arithmetic expressions.

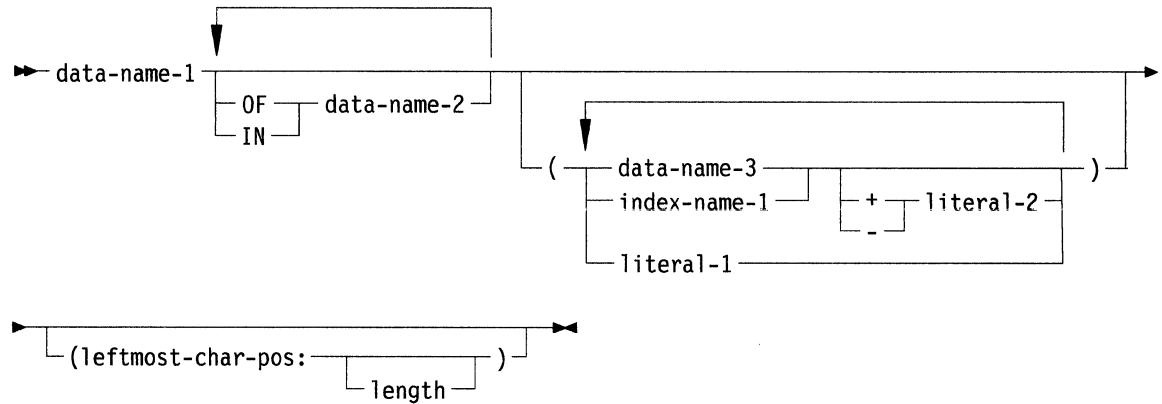
The rules for reference modification are as follows:

1. Each character of a data item referenced by data-name 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 contains a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number within that data item.
2. If the data item referenced by data-name is described as numeric, numeric-edited, alphabetic, or alphanumeric-edited, it is operated upon for reference modification as if it were redefined as an alphanumeric data item of the same size as the data item referenced by data-name.
3. Reference modification for an operand is evaluated as follows:
 - a. If subscripting is specified for the operand, the reference modification is evaluated immediately after evaluation of the subscripts.
 - b. If subscripting is not specified for the operand, the reference modification is evaluated at the time subscripting would have been evaluated if subscripts had been specified.
4. Reference modification creates a unique data item which is a subset of the data item referenced by data-name. This unique data item is defined as follows:
 - a. The evaluation of leftmost-character-position specifies the ordinal position of the leftmost character of the unique data item in relation to the leftmost character of the data item referenced by data-name. Evaluation of leftmost-character-position must result in a positive nonzero integer less than or equal to the number of characters in the data item referenced by data-name.
 - b. The evaluation of length specifies the size of the data item to be used in the operation, in bytes. The evaluation of length must result in a positive nonzero integer. The sum of leftmost-character-position and length minus the value one, must be less than, or equal to, the number of characters in the data item referenced by data-name. If length is not specified, the unique data item 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.
5. The unique data item is considered an elementary data item without the JUSTIFIED clause. It has the same class and category defined for the data item referenced by data-name except that the categories numeric, numeric-edited, and alphanumeric-edited are considered class and category alphanumeric.

Identifier

An identifier is a term used to show that a data-name, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts, or indexes necessary to ensure uniqueness.

The following figure shows the general format for identifiers:



Restrictions on subscripting and indexing are:

1. A data-name must not itself be subscripted nor indexed when that data-name is being used as an index or subscript.
2. Indexing is not permitted where subscripting is not permitted.
3. An index may be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values for index names as data. The form in which these values are stored is dependent on a system-controlling directive. Refer to the *User's Guide* for details. Such data items are called index data items.
4. In the format above, literal-1 must be a positive numeric integer. Literal-2 must be an unsigned numeric integer.

Condition Name

Each condition name must be unique, or be made unique through qualification, indexing, or subscripting. 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 or the conditional variable itself must be used to make the condition name unique.

If references to a conditional variable require indexing or subscripting, then references to any of its condition names also require the same combination of indexing or subscripting.

The format and restrictions on the combined use of qualification, subscripting, and indexing of condition names are exactly those of identifier, except that data-name-1 is replaced by condition-name-1.

In the general formats, condition name refers to a condition name qualified, indexed, or subscripted, as necessary.

Explicit and Implicit Specifications

Four types of explicit and implicit specifications occur in COBOL source programs:

1. Explicit and implicit Procedure Division references
2. Explicit and implicit transfers of control
3. Explicit and implicit attributes
4. Explicit and implicit scope terminators.

Explicit And Implicit Procedure Division References

A COBOL source program can reference data items either explicitly or implicitly in Procedure Division statements. An explicit reference occurs when the name of the referenced item is written in a Procedure Division statement or when the name of the referenced item is copied into the Procedure Division by the processing of a COPY statement. An implicit reference occurs when the item is referenced by a Procedure Division statement without the name of the referenced item being written in the source statement.

An implicit reference also occurs during the execution of a PERFORM statement, when the index or data item referenced by the index name or identifier specified in the VARYING, AFTER or UNTIL phrase is initialized, modified, or evaluated by the control mechanism for the PERFORM statement. Such an implicit reference occurs only if the data item contributes to the execution of the statement.

Explicit and Implicit Transfers of Control

The mechanism controlling program flow transfers control from statement to statement in the sequence in which they were written in the source program, unless an explicit transfer of control overrides this sequence or there is no next executable statement to which control can be passed. The transfer of control from statement to statement occurs without writing an explicit Procedure Division statement and, therefore, is an implicit transfer of control.

COBOL provides both explicit and implicit means of altering the implicit control transfer mechanism.

In addition to the implicit transfer of control between consecutive statements, implicit transfer of control also occurs when the normal flow is altered without the execution of a procedure branching statement. COBOL provides the following types of implicit control flow alterations which override the statement-to-statement transfers of control:

1. When a paragraph is executed under control of another COBOL statement (for example, PERFORM, USE, SORT, and MERGE) control is passed from the last statement of the paragraph, to the control mechanism of the last executed controlling statement. When a paragraph is executed under the control of a PERFORM statement which executes iteratively, and the paragraph is the first paragraph in the range of that PERFORM statement, an implicit transfer of control occurs between the control mechanism for that PERFORM statement and the first statement in the paragraph for each iterative execution of the paragraph.
2. When a SORT or MERGE statement is executed, an implicit transfer of control occurs to any associated input or output procedures.
3. When any COBOL statement is executed which results in the execution of a declarative section, an implicit transfer of control to the declarative section occurs. Note that another implicit transfer of control occurs after execution of the declarative section, as described in 1.

-
4. *In any file operation (including OPEN and CLOSE), if a file does not have a FILE STATUS data item declared for it and the file is not explicitly covered by a USE statement, then it is covered by an implicit USE statement. The implied USE procedure is equivalent to:*

MF

```
USE AFTER ERROR PROCEDURE ON filename  
  
IF status-key-1 = 9  
    DISPLAY error-message UPON CONSOLE  
    STOP RUN.
```

Refer to the User's Guide for the definition of error messages

An explicit transfer of control alters the implicit control transfer mechanism by the execution of a procedure branching or conditional statement. Refer to "Statements and Sentences" on page 2-46. An explicit transfer of control can be caused only by the execution of a procedure branching or conditional statement. The execution of the procedure branching statement ALTER does not in itself constitute an explicit transfer of control, but affects the explicit transfer of control that occurs when the associated GO TO statement is executed. The procedure branching statement EXIT PROGRAM causes an explicit transfer of control when the statement is executed in a called program.

In this document, *next executable statement* is used to refer to the next COBOL statement to which control is transferred according to the rules above and the rules for each language element in the Procedure Division.

There is no next executable statement following:

1. 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.
2. The last statement in a program when the paragraph in which it appears is not being executed under the control of some other COBOL statement.

Explicit and Implicit Attributes

Attributes may be implicitly or explicitly specified. Any attribute which has been explicitly specified is called an explicit attribute. If an attribute has not been specified explicitly, then the attribute takes on the default specification. Such an attribute is known as an implicit attribute.

For example, the usage of a data item need not be specified, in which case a data item's usage is DISPLAY.

Explicit and Implicit Scope Terminators

Scope terminators delimit the scope of certain Procedure Division statements (delimited scope statements) and are of two types: explicit and implicit.

The explicit scope terminators are listed below in Table 2-10 with their matching delimited scope statements. In some cases, the delimited scope statement with which an explicit scope delimiter is paired is determined differently for different COBOL language specifications.

Explicit Scope Terminator	Delimited Scope Statement	
	ANS85	VSC2
END-ADD	ADD	(ADD...) ON SIZE ERROR
END-CALL	CALL	(CALL...) ON OVERFLOW
END-COMPUTE	COMPUTE	(COMPUTE...) ON SIZE ERROR
END-DELETE	DELETE	(DELETE...) INVALID KEY
END-DIVIDE	DIVIDE	(DIVIDE...) ON SIZE ERROR
END-EVALUATE	EVALUATE	EVALUATE
END-IF	IF	IF
END-MULTIPLY	MULTIPLY	(MULTIPLY...) ON SIZE ERROR
END-PERFORM	in-line PERFORM	in-line PERFORM
END-READ	READ	(READ...) AT END and (READ...) INVALID KEY
END-RECEIVE	RECEIVE	
END-RETURN	RETURN	RETURN
END-REWRITE	REWRITE	(REWRITE...) INVALID KEY
END-SEARCH	SEARCH	SEARCH
END-START	START	(START...) INVALID KEY
END-STRING	STRING	(STRING...) ON OVERFLOW
END-SUBTRACT	SUBTRACT	(SUBTRACT...) ON SIZE ERROR
END-UNSTRING	UNSTRING	(UNSTRING...) ON OVERFLOW
END-WRITE	WRITE	(WRITE...) INVALID KEY

Implicit scope termination occurs:

- At the end of any sentence where the separator period terminates the scope of all previous statements not yet terminated.
- Within any statement containing another statement. The next phrase of the containing statement following the contained statement (for example, ELSE, WHEN, AT END, and so on) terminates the scope of any unterminated contained statement.

Program Structure

An AIX VS COBOL program consists of four divisions:

- Identification Division — An identification of the program.
- Environment Division — A description of the equipment to be used to compile and run the program.
- Data Division — A description of the data to be processed.
- Procedure Division — A set of procedures to specify the operations to be performed on the data.

Each division is divided into sections that are further divided into paragraphs, that in turn are made up of sentences.

Any division may be optionally omitted.

MF

Optional Division, Section, and Paragraph Headings

Some of the red tape statements required by the ANSI Standard COBOL Specifications are optional when using AIX VS COBOL. However, it is possible to have AIX VS COBOL system output warning messages when such statements are found to be missing by use of the FLAG directive. Refer to the *User's Guide*. Such statements are identified as optional in this manual by enclosing them between brackets and highlighting them. The symbols next to the highlighted areas indicate the dialect in which these features are optional.

Identification Division

This section describes the organization, structure, and general format of the Identification Division.

The Identification Division must be included in every COBOL source program except ANSI 85, where it is optional. This division identifies both the source program and the resultant output listing. In addition, the user may include the date the program is written, the date of the compilation of the source program, and other information as desired under the paragraphs in the general format shown in "General Format."

Organization

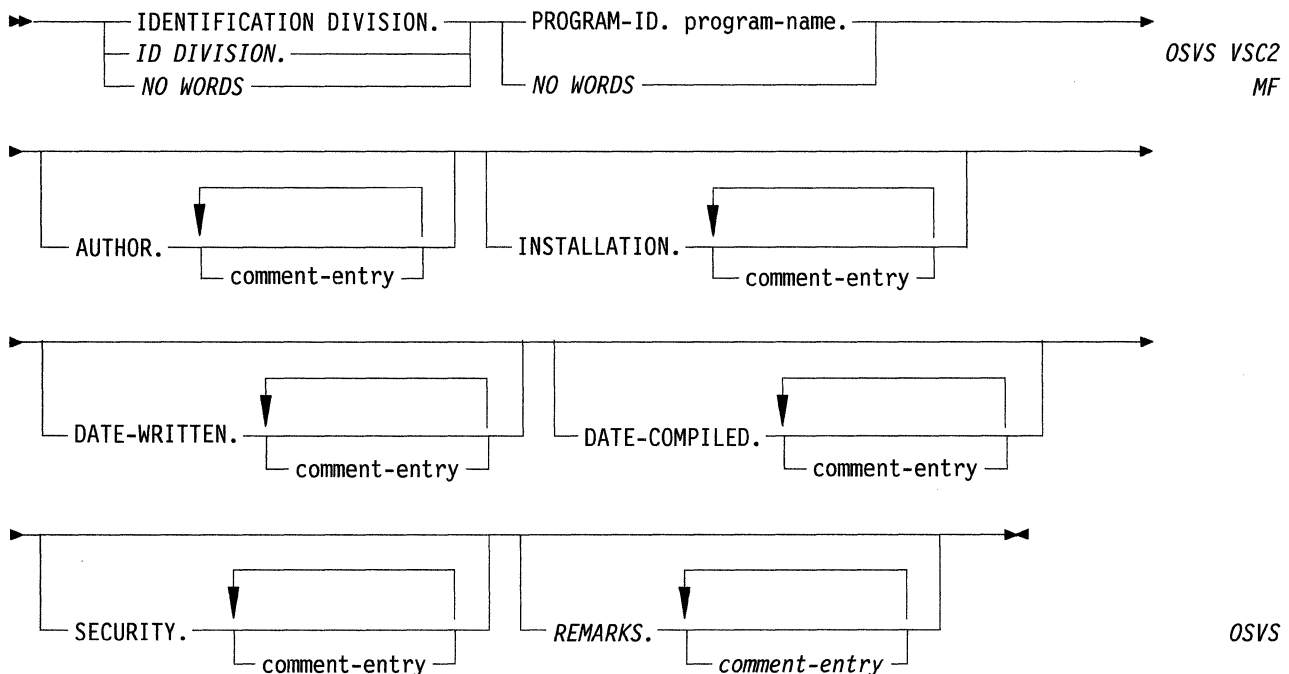
Paragraph headers identify the type of information contained in the paragraph. The name of the program must be specified in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be included in this division at the user's discretion, in the order of presentation shown by the format in "General Format."

Structure

The general format of the paragraphs in the Identification Division defines the order of presentation in the source program.

General Format

The following figure shows the general format of the paragraphs in the Identification Division:



Environment Division

This section describes the organization, structure, and general format of the Environment Division.

The Environment Division specifies a standard method of expressing those aspects of a data processing problem dependent upon the physical characteristics of a specific computer. This division allows specification of the configuration of the source computer and the object computer. In addition, information relating to input-output control, special hardware characteristics, and control techniques can be specified in this division.

The Environment Division must be included in every COBOL source program.

Organization

Two sections make up the Environment Division:

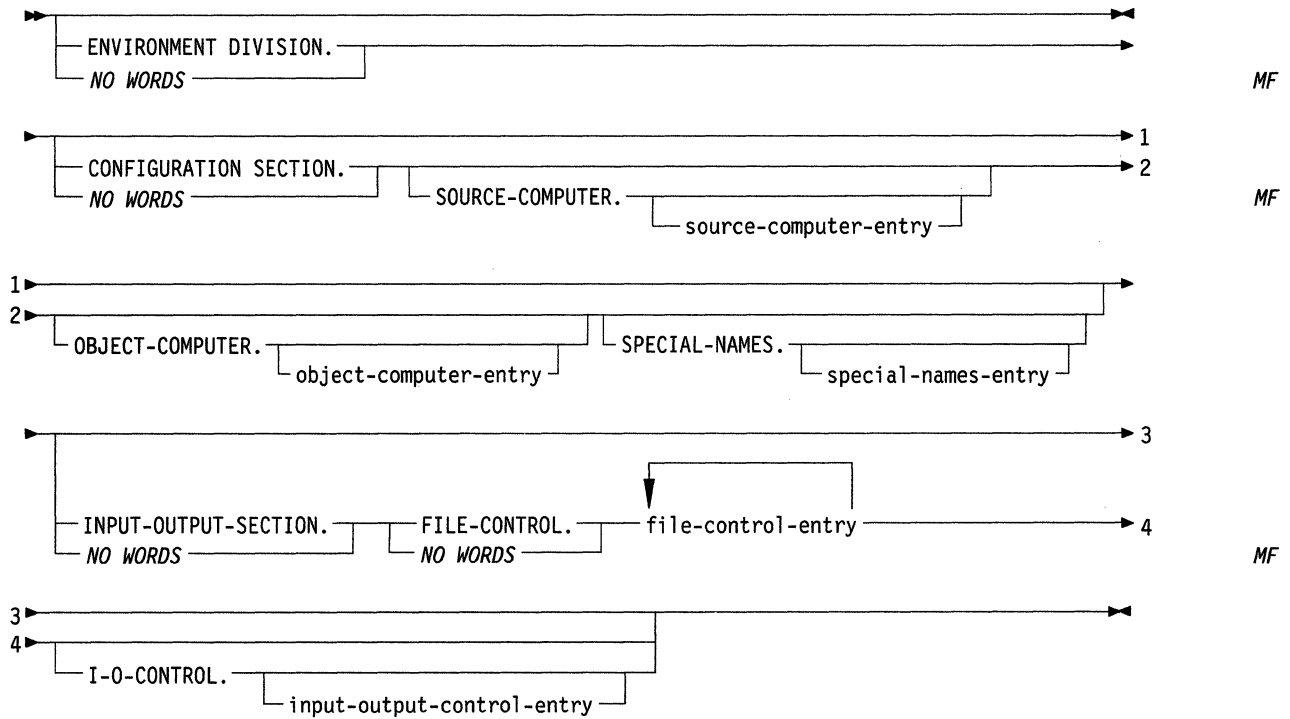
- The Configuration Section deals with the characteristics of the source computer and the object computer. This section is divided into three paragraphs:
 - The SOURCE-COMPUTER paragraph, describing the computer configuration on which the intermediate code is produced
 - The OBJECT-COMPUTER paragraph, describing the computer configuration on which the object (intermediate code) program is to be run
 - The SPECIAL-NAMES paragraph, that relates the implementation names used by the AIX VS COBOL system to the mnemonic names used in the source program.
- The INPUT-OUTPUT SECTION deals with the information needed to control transmission and handling of data between external media and the object program. This section is divided into two paragraphs:
 - The FILE-CONTROL paragraph, which names and associates the files with external media
 - The I-O-CONTROL paragraph, which defines special control techniques to be used in the object program.

Structure

The general format of the sections and paragraphs in the Environment Division defines the order of presentation in the source program.

General Format

The following figure shows the general format of the sections and paragraphs in the Environment Division:



Data Division

This section defines the physical and logical aspects of data description.

The Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output.

Data Division Organization

The Data Division is divided into sections. These are the FILE, WORKING-STORAGE, LINKAGE, COMMUNICATION, REPORT and SCREEN SECTIONS. *MF*

The FILE SECTION defines the structure of data files. Each file is defined by a file description entry and one or more record descriptions. Record descriptions are written immediately following the file description entry.

The WORKING-STORAGE SECTION describes records and noncontiguous data items that are not part of external data files but are developed and processed internally. It also describes data items whose values are assigned in the source program and do not change during the execution of the object program.

The LINKAGE SECTION appears in the called program and describes data items to be referred to by the calling and the called program. Its structure is the same as the WORKING-STORAGE SECTION.

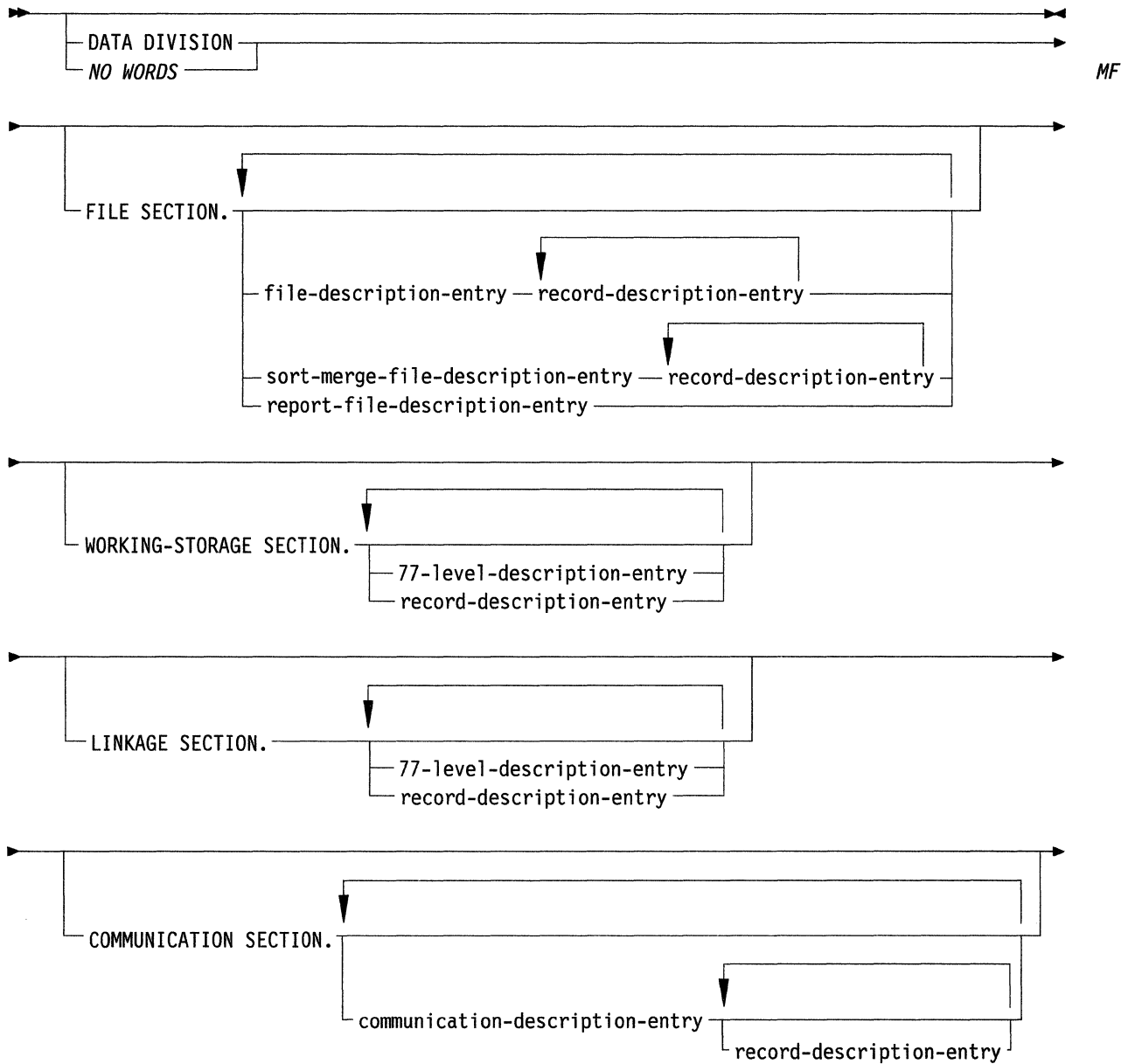
The COMMUNICATION SECTION describes the data item in the source program that serves as the interface between the MCS and the program.

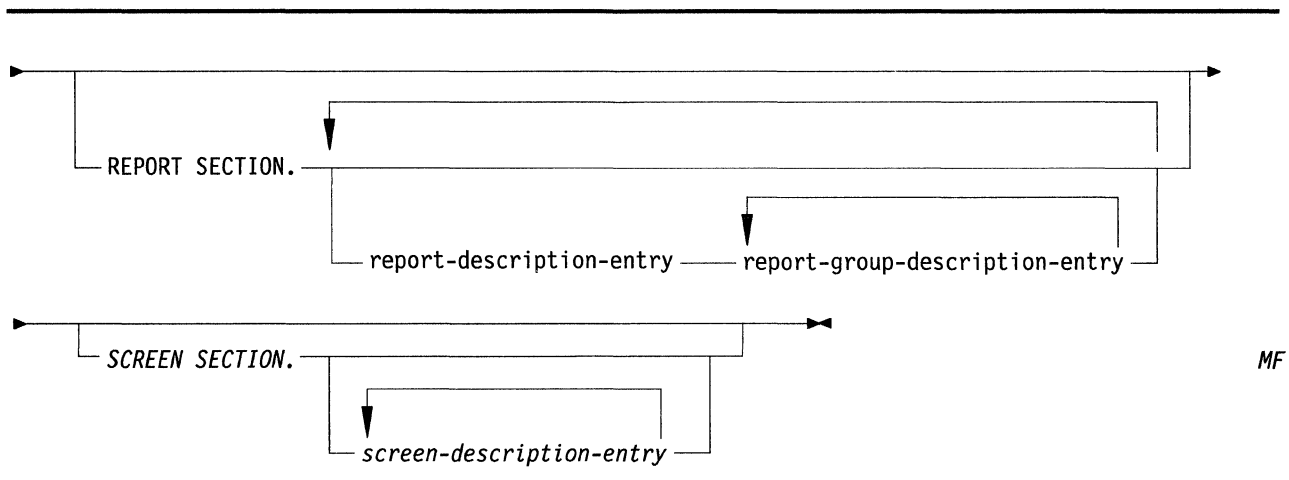
The REPORT SECTION contains one or more report description entries (RD entries), each of which forms the complete description of a report.

The SCREEN SECTION defines the attributes of the display screens. It specifies the exact location of fields when they are displayed on the screen and controls certain console features during an ACCEPT or DISPLAY operation. *MF*

General Format

The following figure shows the general format of the sections in the Data Division and defines the order of their presentation in the source program:





MF

Procedure Division

This section describes the statements and sentences components of the Procedure Division. The Procedure Division may contain declarative and nondeclarative procedures.

Declaratives

Declarative sections must be grouped at the beginning of the Procedure Division preceded by the key word `DECLARATIVES` and followed by the key words `END DECLARATIVES` (refer to “USE Statement” on page 8-86).

Procedures

A procedure is composed of a paragraph, a group of successive paragraphs, a section, or a group of successive sections within the Procedure Division. A procedure name is a word used to refer to a paragraph or section in the source program in which it occurs. It consists of a paragraph name (which may be qualified) or a section name.

The end of the Procedure Division and the physical end of the program is either the physical position in a COBOL source program after which no further procedures appear, or the occurrence of the `END PROGRAM` header.

A section consists of a section header followed by zero, one, or more successive paragraphs. A section ends immediately before the next section, at the end of the Procedure Division, or at the key words `END DECLARATIVES` in the declaratives portion of the Procedure Division.

A paragraph consists of a paragraph name followed by a period and a space, and by zero, one, or more successive sentences. A paragraph ends immediately before the next paragraph name or section name, at the end of the Procedure Division, or at the key words `END DECLARATIVES` in the declaratives portion of the Procedure Division.

A sentence consists of one or more statements and is terminated by a period followed by a space.

A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb.

The term identifier is defined as the word or words necessary to make unique reference to a data item.

Execution

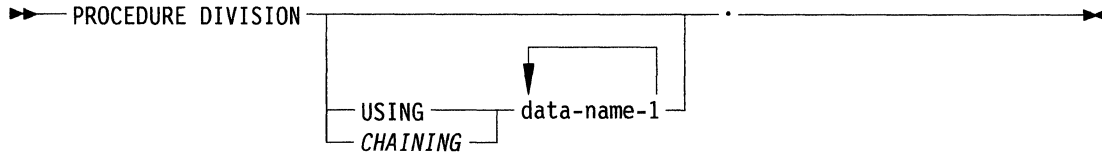
Execution begins with the first statement of the Procedure Division, excluding declaratives. Statements are then executed in the order in which they occur in the source program, except where the rules indicate some other order.

General Format

The following figures show the general format for the Procedure Division:

PROCEDURE DIVISION Header

The Procedure Division is identified by and must begin with the following header:

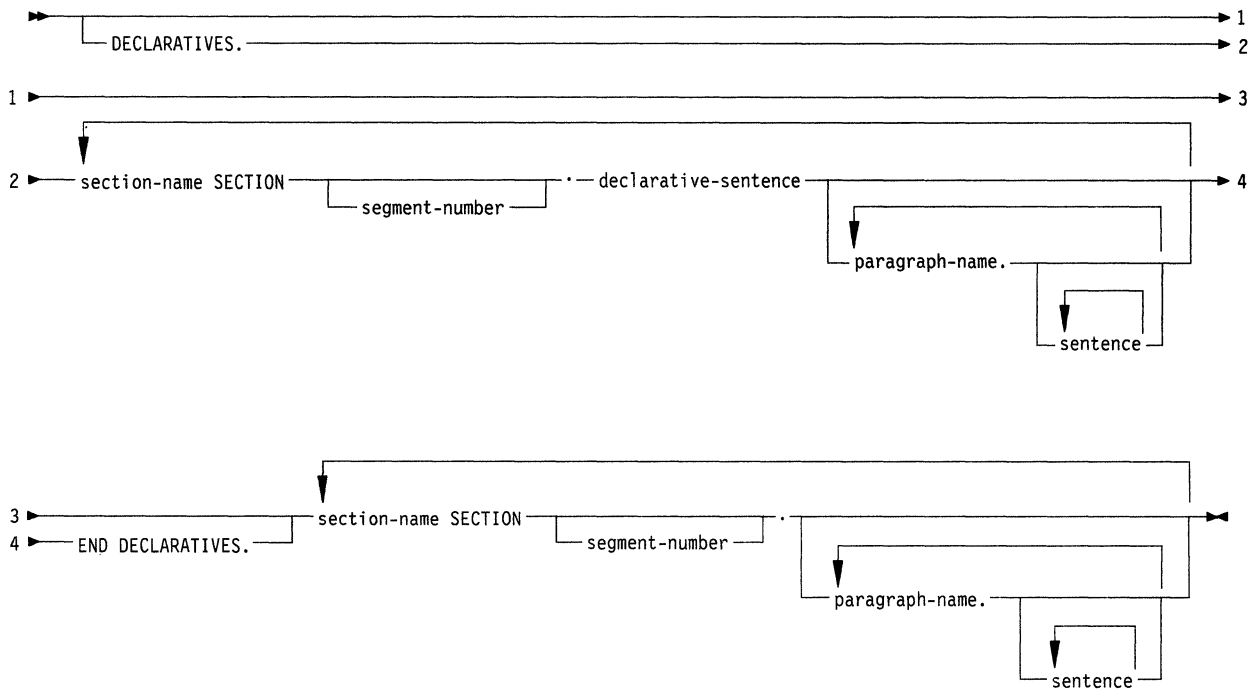


MF

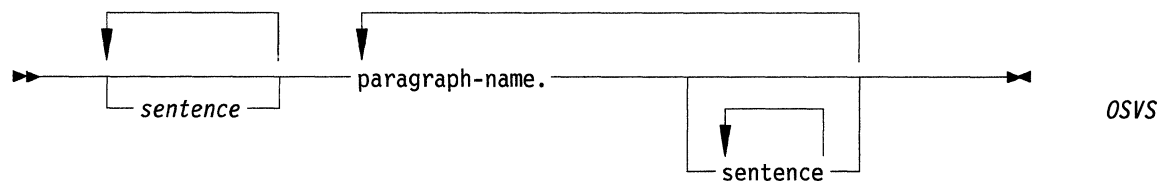
PROCEDURE DIVISION Body

The body of the Procedure Division must conform to one of the following formats:

Format 1



Format 2



OSVS

Statements and Sentences

There are four types of statements:

- Conditional statements
- AIX VS COBOL system directing statements
- Imperative statements
- Delimited scope statements.

There are three types of sentences:

- Conditional sentences
- AIX VS COBOL system directing sentences
- Imperative sentences.

Conditional Statement

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.

A conditional statement is one of the following:

- An EVALUATE, IF, SEARCH, or RETURN statement
- A READ statement specifying the AT END or INVALID KEY phrase
- *An ON statement* OSVS
- A WRITE statement specifying the INVALID KEY or END-OF-PAGE phrase
- A START, REWRITE, or DELETE statement specifying the INVALID KEY PHRASE
- An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) specifying the SIZE ERROR phrase
- A RECEIVE statement specifying a NO DATA phrase
- A STRING or UNSTRING statement specifying the ON OVERFLOW phrase
- A CALL statement specifying the ON OVERFLOW or ON EXCEPTION phrase.

Conditional Sentence

A conditional sentence is a conditional statement, optionally preceded by an imperative statement, terminated by a period followed by a space.

AIX VS COBOL Directing Statement

An AIX VS COBOL system directing statement consists of a directing verb and its operands. The directing verbs are:

<i>COPY</i>	
<i>ENTER</i>	
<i>USE</i>	
<i>EJECT</i>	<i>OSVS VSC2</i>
<i>SKIP1</i>	
<i>SKIP2</i>	
<i>SKIP3</i>	
<i>TITLE</i>	<i>VSC2</i>

An AIX VS COBOL system directing statement causes the AIX VS COBOL system to take a specified action during creation of the object code.

AIX VS COBOL System Directing Sentence

An AIX VS COBOL system directing sentence is a single directing statement, terminated by a period, followed by a space.

Imperative Statement

An imperative statement indicates a specific unconditional action to be taken by the object program. An imperative statement is any statement that is neither a conditional statement nor an AIX VS COBOL system directing statement. An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator.

The imperative verbs are:

<i>ACCEPT</i>	<i>ENABLE</i>	<i>RECEIVE⁴</i>
<i>ADD¹</i>	<i>EXIT</i>	<i>RELEASE</i>
<i>ALTER</i>	<i>GO TO</i>	<i>REWRITE²</i>
<i>CALL³</i>	<i>INITIALIZE</i>	<i>SEND</i>
<i>CANCEL</i>	<i>INSPECT</i>	<i>SET</i>
<i>CLOSE</i>	<i>MERGE</i>	<i>SORT</i>
<i>COMPUTE¹</i>	<i>MOVE</i>	<i>START²</i>
<i>CONTINUE</i>	<i>MULTIPLY¹</i>	<i>STOP</i>
<i>DELETE²</i>	<i>OPEN</i>	<i>STRING³</i>
<i>DISABLE</i>	<i>PERFORM</i>	<i>SUBTRACT¹</i>
<i>DISPLAY</i>	<i>PURGE</i>	<i>UNSTRING³</i>
<i>DIVIDE¹</i>	<i>READ⁵</i>	<i>WRITE⁶</i>

Notes:

1. Without the optional *SIZE ERROR* phrase.
2. Without the optional *INVALID KEY* phrase.
3. Without the optional *ON OVERFLOW* phrase.
4. Without the optional *NO DATA* phrase.
5. Without the optional *AT END* phrase or *INVALID KEY* phrase.
6. Without the optional *INVALID KEY* phrase or *END-OF-PAGE* phrase.

The OSVS imperative verbs are:

- *EXAMINE* OSVS
- *EXHIBIT*
- *GOBACK*
- *TRANSFORM*

The VSC2 imperative verb is:

- *GOBACK* VSC2

The Micro Focus imperative verb is:

- *EXEC(UTE)* MF

The term “imperative statement” in the general format of statements, refers to the sequence of imperative statements that must end with a period or an ELSE phrase for a previous IF statement.

Delimited Scope Statements

A delimited scope statement is a statement (for example, an IF statement) which is terminated by a matching explicit scope terminator (in this case, END-IF). Thus, all the statements between a delimited scope statement and its paired explicit scope terminator are deemed to be contained within that delimited scope statement.

Delimited scope statements may be nested, in which case each explicit scope terminator encountered in the program is considered to pair with the nearest preceding unpaired matching delimited scope statement.

Delimited scope statements may also be implicitly terminated. They may be terminated at the end of a procedural sentence (where all nonterminated statements are terminated by the separator period). Scope delimited statements may also be terminated by the termination of any containing delimited scope statement. For example, an ELSE phrase of a containing IF statement terminating a contained in-line PERFORM statement without the presence of an END-PERFORM phrase.

Note that not all statements are scope-delimitable in this fashion. Statements that are scope-delimitable are only termed delimited scope statements if they are explicitly terminated by an explicit scope delimiter.

See “Explicit and Implicit Scope Terminators” on page 2-36 for more information.

Imperative Sentence

An imperative sentence is an imperative statement terminated by a period followed by a space.

Categories of Statements

The following figures list categories and verbs:

Category	Verbs	
Arithmetic	{ ADD COMPUTE DIVIDE EXAMINE (TALLYING) INSPECT (TALLYING) MULTIPLY SUBTRACT	OSVS
Conditional	{ ADD (SIZE ERROR) CALL (OVERFLOW) COMPUTE (SIZE ERROR) DELETE (INVALID KEY) DIVIDE (SIZE ERROR) EVALUATE GO TO (DEPENDING) IF MULTIPLY (SIZE ERROR) ON READ (END or INVALID KEY) RECEIVE (NO DATA) RETURN (END) REWRITE (INVALID KEY) SEARCH START (INVALID KEY) STRING (OVERFLOW) SUBTRACT (SIZE ERROR) UNSTRING (OVERFLOW) WRITE (INVALID KEY or END-OF-PAGE)	OSVS

Category	Verbs	
Data Movement	ACCEPT (DATE, DAY or TIME)	
	ACCEPT MESSAGE COUNT	
	<i>EXAMINE</i>	<i>OSVS</i>
	INITIALIZE	
	INSPECT (CONVERTING)	
	INSPECT (REPLACING)	
	MOVE	
	STRING	
Ending	<i>TRANSFORM</i>	<i>OSVS</i>
	UNSTRING	
Ending	<i>GOBACK</i>	<i>OSVS VSC2</i>
	STOP	
Input-Output	ACCEPT (identifier)	
	CLOSE	
	DELETE	
	DISABLE	
	DISPLAY	
	ENABLE	
	<i>EXHIBIT</i>	<i>OSVS</i>
	OPEN	
	PURGE	
	READ	
	RECEIVE	
	REWRITE	
	SEND	
	SET(TO ON or TO OFF)	
	START	
	STOP (literal)	
WRITE		

Category	Verbs	
Inter-Program Communication	{ CALL CANCEL	
Null Operation	{ CONTINUE EXIT	
Ordering	{ MERGE RELEASE RETURN SORT	
Procedure Branching	{ ALTER CALL EXIT GO TO PERFORM	
Scope Delimiting	{ END-ADD END-CALL END-DELETE END-DIVIDE END-EVALUATE END-IF END-MULTIPLY END-PERFORM END-READ END-RECEIVE END-RETURN END-REWRITE END-SEARCH END-START END-STRING END-SUBTRACT END-UNSTRING END-WRITE	
VS COBOL system Directing	{ COPY EJECT ENTER ENTRY EXECUTE SKIP1 SKIP2 SKIP3 TITLE USE	OSVS VSC2 OSVS VSC2 MF OSVS VSC2 VSC2
Table Handling	{ SEARCH SET	

Reference Format

The reference format provides a standard method for describing COBOL source programs.

The Reference format is described in terms of character positions in a line on an input-output medium. The AIX VS COBOL system accepts source programs written in reference format and produces an output listing of the source program input in reference format. See Chapter 1, "Introduction" for a sample source program.

The rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing.

The divisions of a source program must be ordered as follows:

1. Identification Division
2. Environment Division
3. Data Division
4. Procedure Division.

Each division must be written according to the rules for the reference format.

Reference Format Representation

The reference format for a line is represented as in Figure 2-6.

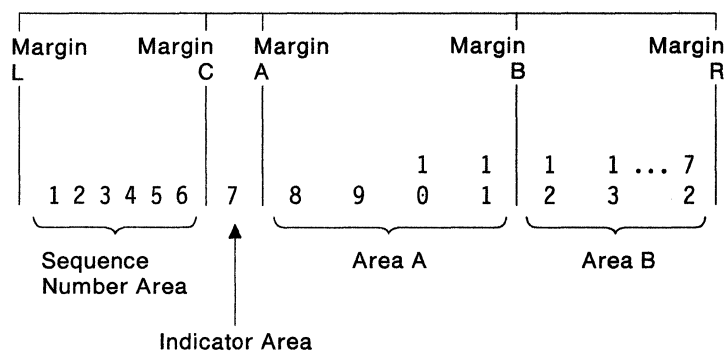


Figure 2-6. Reference Format for a COBOL Source Line

- Margin L is immediately to the left of the leftmost character position of a line.
- Margin C is between the 6th and 7th character positions of a line.
- Margin A is between the 7th and 8th character positions of a line.
- Margin B is between the 11th and 12th character positions of a line.
- Margin R is immediately to the right of the 72nd character position of a line.

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

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

Area A occupies character positions 8, 9, 10 and 11 and is between Margin A and Margin B.

Area B occupies character positions 12 through 72 inclusive; it begins immediately to the right of Margin B and terminates immediately to the left of Margin R.

Sequence Numbers

A sequence number, normally consisting of six digits, may be placed in the sequence area and may be used to label a source program line. This sequence number is usually in ascending numeric order for each successive source statement line of the program.

The ascending order of sequence numbers may be optionally verified by the AIX VS COBOL system. Refer to the *User's Guide* for details of the directive that enables this.

There is no requirement for the content of this area to be numeric, or even unique.

If the first character position of the sequence number field contains an asterisk, or any unprintable control character (less than character SPACE in the ASCII collating sequence), the line is treated as source-comment and is ignored and not output to the listing file or device. *MF*

Continuation of Lines

Whenever a sentence, entry, phrase, or clause requires more than one line, it may be continued by starting subsequent line(s) in area B. These subsequent lines are called continuation lines. The line being continued is called the continued line. Any word or literal may be broken in such a way that part of it appears on a continuation line.

A hyphen in the indicator area of a line indicates that the first nonblank character in area B of the current line is the successor of the last nonblank character of the preceding line without any intervening space. However, if the continued line contains a nonnumeric literal without a closing quotation mark, the first nonblank character in area B on the continuation line must be a quotation mark. The continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal. Area A of a continuation line must be blank.

If there is no hyphen in the indicator area of a line, it is assumed that a space follows the last character in the preceding line. Both characters composing the separator == must be on the same line.

DBCS Support

DBCS literals and mixed literals may not be continued.

End of DBCS Support

Blank Lines

A blank line is one that is blank from Margin C to Margin R, inclusive. A blank line can appear anywhere in the source program, except immediately preceding a continuation line.

Pseudo-Text

The character strings and separators comprising pseudo-text may start in either area A or area B. However, if 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. The normal rules for continuation of lines apply to the formation of text words. (See Chapter 8, "File Input and Output.")

Division, Section, and Paragraph Formats

The following are the Division, Section, and Paragraph formats.

Division Header

The division header must start in area A.

Section Header

The section header must start in area A.

A section consists of zero or more paragraphs in the Environment and Procedure Division and zero or more Data Division entries in the Data Division.

Paragraph Header, Paragraph Name, and Paragraph

A paragraph consists of a paragraph name followed by a period and a space, and by zero, one or more sentences, or a paragraph header followed by one or more entries. Comment entries may be included within a paragraph. The paragraph header or paragraph name starts in area A of any line following the first line of a division or a section.

The first sentence or entry in a paragraph begins either on the same line as the paragraph header or paragraph name, or in area B of the next nonblank line that is not a comment line. Successive sentences or entries begin either 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.

Note: *In AIX VS COBOL programs, sentences may begin anywhere in area A or area B.* *MF*

When sentences or entries of a paragraph require more than one line, they may be continued as described in “Continuation of Lines” on page 2-53 in this chapter.

Data Division Entries

Each Data Division entry begins with a level indicator or a level number, followed by a space, followed by its associated name (except in the REPORT SECTION), followed by a sequence of independent descriptive clauses. Each clause, except the last clause of an entry, may be terminated by either the separator semicolon or the separator comma. The last clause is always terminated by a period followed by a space.

There are two types of Data Division entries: those which begin with a level indicator and those which begin with a level number.

A level indicator is any of the following:

- FD (see “File Description – Complete Entry Skeleton” on page 8-35)
- SD (see “Screen Description – Complete Entry Skeleton” on page 18-15)
- CD (see “Communication Description – Complete Entry Skeleton” on page 15-4)
- RD (see “Data Division in the Report Writer Module” on page 14-11).

In those Data Division entries that begin with a level indicator, the level indicator begins in area A followed by a space and followed with its associated name and appropriate descriptive information in area B.

Those Data Division entries that begin with level-numbers are called data description entries.

A level number has a value taken from the set of values 1 through 49, 66, 77, 78, and 88. *MF*

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. At least one space must separate a level number from the word following the level-number.

In those Data description entries that begin with level-number 01 or 66, 77, 78, and 88, the level-number begins in area A. It is followed by a space, which is followed by its associated record name or item name and appropriate descriptive information in area B.

MF

Successive data description entries may have the same format as the first or may be indented according to level number. The entries in the output listing require indentation only if the input is indented. Indentation does not affect the magnitude of a level number.

When level numbers are to be indented, each new level number may begin any number of spaces to the right of margin A. The extent of indentation to the right is determined only by the width of the physical medium.

Declaratives

The key word DECLARATIVES and the key words END DECLARATIVES indicate the beginning and end of the declaratives portion of the Procedure Division. The key word DECLARATIVES and the key words END DECLARATIVES must each appear on a line by themselves. Each must begin in area A and be followed by a period and a space.

Comment Lines

A comment line is any line with an asterisk (*) in the continuation indicator area of the line. *A comment line can appear as any line in a source program* after the Identification Division header. Any combination of characters from the character set may be included in area A and area B of that line. The asterisk and the characters in area A and area B will be produced on the listing but serve as documentation only.

MF

A second form of comment line represented as above but with a slash (/) (instead of an asterisk) in the indicator area of the line causes page ejection prior to printing the comment.

Successive comment lines are allowed. Continuation of comment lines is permitted, except that each continuation line must contain an * in the indicator area.

Reserved Words

A full list of reserved words is given in Appendix D, "Reserved Word List."

PART 2. The Nucleus



Language Reference

Chapter 3. Introduction to the Nucleus

Contents

About This Chapter	3-3
Function of the Nucleus	3-4
Overall Language	3-4
Name Characteristics	3-4
Figurative Constants	3-4
Reference Format	3-4
Subscripting	3-4
A COBOL Source Program	3-5
Organization	3-5
Structure	3-5
End Program Header	3-6

About This Chapter

The nucleus module provides a basic language capability for the internal processing of data within the structure of the four divisions of a COBOL program. The nucleus also provides table defining and accessing capabilities and a debugging capability.

Function of the Nucleus

The nucleus provides a basic language capability for the internal processing of data within the basic structure of the four divisions of a program.

Overall Language

This section describes elements that are common to the entire language such as the name characteristics, figurative constants, reference format, and subscripting levels of the nucleus.

Name Characteristics

IBM AIX VS COBOL data-names need not begin with an alphabetic character. Alphabetic characters may be positioned anywhere within the data-name. Qualification is permitted and all data-names, condition names, paragraph names, and text names need not be unique, provided that all references are sufficiently qualified to be unambiguous.

Figurative Constants

The following figurative constants may be used: ZERO, ZEROS, ZEROES, SPACE, SPACES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, LOW-VALUES, NULL, NULLS, QUOTE, QUOTES, ALL literal, and ALL figurative constant. VSC2

Reference Format

A word, numeric literal, or PICTURE character string can be broken in such a way that part of it appears on a continuation line.

Subscripting

Up to 16 levels of subscriptions are permitted.

A COBOL Source Program

This section describes the organization and structure of a COBOL source program. A COBOL source program is a syntactically correct set of COBOL statements.

Organization

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

1. Identification Division
2. Environment Division
3. Data Division
4. Procedure Division.

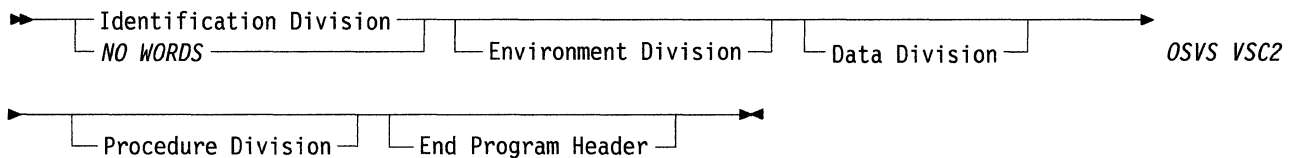
The end of a COBOL source program is indicated by either the End Program header, if specified, or by the absence of further source program lines.

Structure

The following section gives the general format and order of presentation of the entries and statements which constitute a COBOL source program.

General Format

The following figure shows the format of a COBOL source program:



Syntax Rule

The generic terms Identification Division, Environment Division, Data Division, Procedure Division, and End Program Header represent a COBOL Identification Division, a COBOL Environment Division, a COBOL Data Division, a COBOL Procedure Division, and a COBOL End Program Header, respectively.

General Rule

The beginning of a division in a program is indicated by the appropriate division header. The end of a division is indicated by one of the following:

- The division header of a succeeding division in that program
- The physical position after which no more source program lines occur
- The End Program Header.

End Program Header

This section describes the function, general format, syntax, and general rules of the End Program Header.

Function

The End Program Header indicates the end of the named COBOL source program.

General Format

The following figure shows the format of the End Program Header:

▶— END PROGRAM program-name. —▶

Syntax Rules

The following syntax rules apply to the End Program Header:

1. The program name must conform to the rules for forming a user-defined word.
2. The program name must be identical to the program name declared in the PROGRAM-ID paragraph. Refer to “PROGRAM-ID Paragraph” on page 4-6.

General Rules

The following general rules apply to the End Program Header:

1. The End Program Header indicates the end of the specified COBOL source program.
2. If the next source statement after the program terminated by the End Program Header is a COBOL statement, it must be the Identification Division header of a program to be compiled separately from that program terminated by the End Program Header. Refer to “END PROGRAM Header” on page 11-13.

Chapter 4. Identification Division in the Nucleus

Contents

About This Chapter	4-3
General Description	4-4
Organization	4-4
Structure	4-4
General Format	4-4
Syntax Rules	4-5
PROGRAM-ID Paragraph	4-6
Function	4-6
General Format	4-6
Syntax Rules	4-6
General Rules	4-6
DATE-COMPILED Paragraph	4-7
Function	4-7
General Format	4-7
Syntax Rule	4-7
General Rule	4-7
REMARKS Paragraph	4-8

About This Chapter

This chapter describes the Identification Division of a COBOL program.

The Identification Division identifies the program. It must be the first division in a COBOL source program, if specified. It names the program, and it may include the date the program was written, the date of compilation, the author, and other documentary information.

General Description

The Identification Division identifies the source program and its output listing. In addition, you may include the date the program is written and other information under the paragraphs in the general format shown below. This division (including the division header) is optional.

Organization

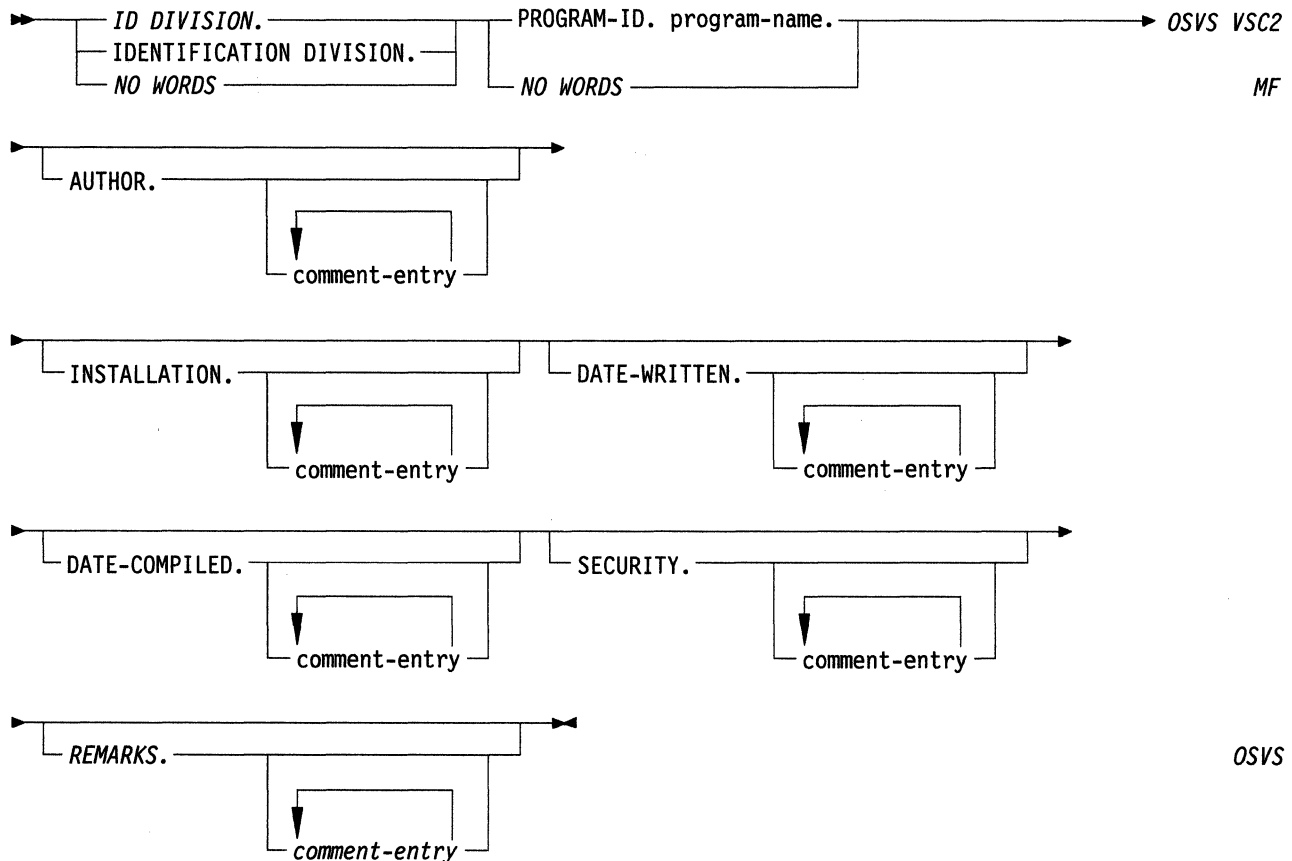
Paragraph headers identify the type of information contained in the paragraph. The name of the program must be given in the PROGRAM-ID paragraph. The other paragraphs are optional and may be included in this division at your discretion, in the order of presentation shown by the general format below.

Structure

The general format of the paragraphs in the Identification Division is given below. *Paragraphs may appear in any order.* OSVS VSC2

General Format

The following figure shows the format of the Identification Division:



Syntax Rules

The following syntax rules apply to the Identification Division:

1. *Paragraphs may appear in any order.* OSVS VSC2
2. *Periods following the paragraph names within the Identification Division are optional.* OSVS VSC2
3. The comment-entry may be any combination of characters from the character set. The continuation of the comment-entry by the use of a hyphen in the continuation area is not permitted; however, the comment-entry may be contained on one or more lines but is restricted to area B of those lines. The next line beginning in area A begins the next noncomment paragraph.

The comment-entry may contain the SKIP1, SKIP2, SKIP3, EJECT, or TITLE statements anywhere on the line. If any of these statements appears alone on a line within the comment-entry, the statement performs its function and does not terminate the comment-entry. OSVS VSC2
VSC2

The comment-entry may be contained in either area A or area B of the comment-entry lines. However, the next occurrence within area A of any one of the following COBOL words or phrases terminates the comment-entry and begins the next paragraph or division: OSVS

*PROGRAM-ID
AUTHOR
INSTALLATION
DATE-WRITTEN
DATE-COMPILED
SECURITY
ENVIRONMENT
DATA
PROCEDURE*

DBCS Support

4. Comments may combine Double-Byte Character Set (DBCS) and Single-Byte Character Set (SBCS) character-strings.
Comment-entries may have DBCS strings in the Identification Division.
Multiple lines are allowed in a comment-entry containing DBCS strings.

End of DBCS Support

PROGRAM-ID Paragraph

Function

The PROGRAM-ID paragraph gives the name by which a program is identified.

General Format

The following shows the general format of the PROGRAM-ID paragraph:

▶▶ PROGRAM-ID. program-name. ◀◀

Syntax Rules

The following syntax rules apply to the PROGRAM-ID paragraph:

1. The program name must conform to the rules for formation of a user-defined word.
2. *The program name may be a nonnumeric literal.* OSVS VSC2

General Rules

The following general rules apply to the PROGRAM-ID paragraph:

1. The program name is associated with the object code file pertaining to this COBOL program.
2. *The PROGRAM-ID paragraph is optional.* MF

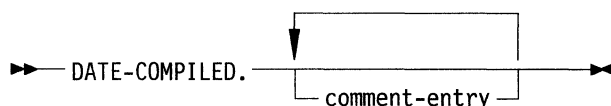
DATE-COMPILED Paragraph

Function

The DATE-COMPILED paragraph provides the date the intermediate code was produced in the Identification Division source program listing.

General Format

The following shows the general format of the DATE-COMPILED paragraph:



Syntax Rule

The comment-entry may be any combination of the characters from the character set. The continuation of the comment-entry by use of the hyphen is not permitted; however, the comment-entry may be contained on one or more lines. The comment-entry lines must be contained within area B. The next line beginning in area A begins the next noncomment paragraph.

General Rule

The paragraph name DATE COMPILED causes the IBM AIX VS COBOL system to insert a date comment-entry as the intermediate code is created. If a DATE-COMPILED paragraph is present (with or without a comment-entry), the paragraph is replaced in the program listing with one of the form:

►— DATE COMPILED. — current date and time. —►

where the date and time format is DD-MM-YYhh:mm.

Refer to the *User's Guide* for details of the derivation of the comment-entry replacement string for your COBOL implementation at the time the intermediate code is produced.

REMARKS Paragraph

*The REMARKS paragraph allows a comment-entry to be specified. See OSVS
syntax rule 3 on page 4-5.*

Chapter 5. Environment Division in the Nucleus

Contents

About This Chapter	5-3
General Description	5-4
Configuration Section	5-4
SOURCE-COMPUTER Paragraph	5-5
Function	5-5
General Format	5-5
Syntax Rules	5-5
General Rules	5-5
OBJECT-COMPUTER Paragraph	5-6
Function	5-6
General Format	5-6
Syntax Rule	5-6
General Rules	5-6
SPECIAL-NAMES Paragraph	5-8
Function	5-8
General Format	5-9
Syntax Rules	5-10
General Rules	5-11
Example	5-14

About This Chapter

The Environment Division has two sections: the Configuration Section and the Input-Output Section. This chapter addresses the Configuration Section.

The Configuration Section optionally does one or more of the following:

- Specifies the computer on which the source program is compiled
- Specifies the computer on which the object program is executed
- Relates IBM-defined environment names to user-defined mnemonic names
- Specifies a substitution for the currency sign
- Interchanges the functions of the comma and the period in numeric PICTUREs and numeric literals
- Relates alphabet names to character sets or collating sequences
- Relates class names to sets of characters.

General Description

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer.

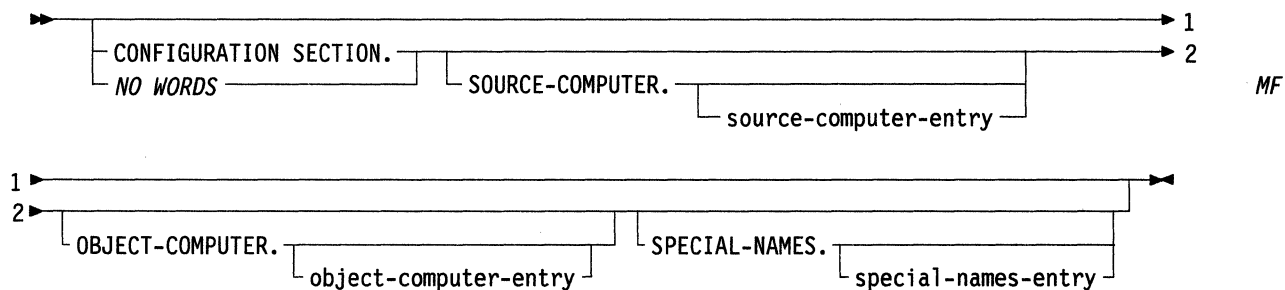
The Environment Division is optional in a COBOL source program.

Configuration Section

The Configuration Section is located in the Environment Division of a source program. The Configuration Section deals with the characteristics of the source computer and the object computer. This section also provides a means for specifying the currency sign, choosing the decimal point, relating implementer names to user-specified mnemonic names, relating alphabet names to character sets or collating sequences, and relating class names to sets of characters.

The Configuration Section is optional in the Environment Division of a COBOL source program. VSC2

The following figure shows the general format of the Configuration Section:



SOURCE-COMPUTER Paragraph

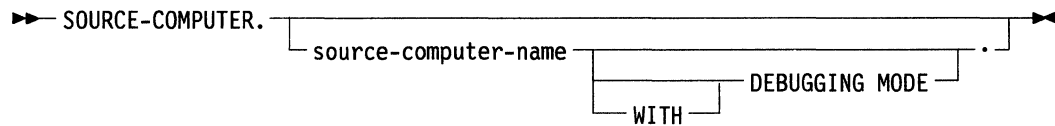
Function

The SOURCE-COMPUTER paragraph identifies the computer upon which the program is to be compiled. *It is optional.*

OSVS VSC2

General Format

The following figure shows the general format for the SOURCE-COMPUTER paragraph:



Syntax Rules

The following syntax rules apply to the SOURCE-COMPUTER paragraph:

1. *source-computer-name* must be one COBOL word defined by the user.
2. The SOURCE-COMPUTER paragraph may consist of only the SOURCE-COMPUTER header.

General Rules

The following general rules apply to the SOURCE-COMPUTER paragraph:

1. The source-computer-name provides a means for identifying equipment configuration, in which case you specify the computer name and its implied configuration. The SOURCE-COMPUTER paragraph is used for documentary purposes only.
2. The WITH DEBUGGING MODE phrase is used to enable debugging code in accordance with Standard ANSI COBOL Debug. Refer to “Environment Division in COBOL Debug” on page 17-5.

OBJECT-COMPUTER Paragraph

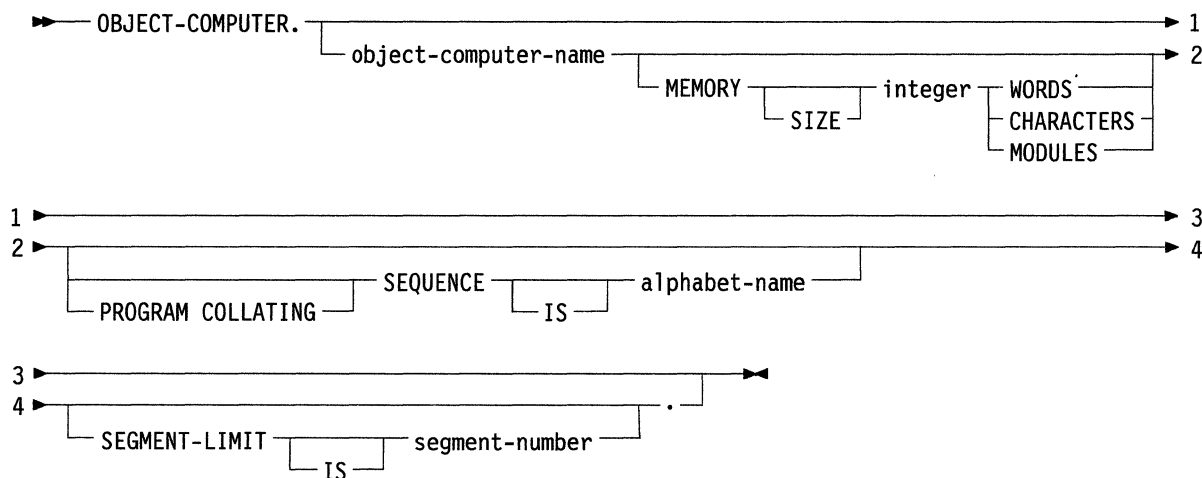
Function

The OBJECT-COMPUTER paragraph identifies the computer on which the program is to be executed. *It is optional.*

OSVS VSC2

General Format

The following figure shows the general format of the OBJECT-COMPUTER paragraph:



Syntax Rule

object-computer-name must be one COBOL word defined by the user.

General Rules

The following general rules apply to the OBJECT-COMPUTER paragraph:

1. The computer name provides a means for identifying equipment configuration, in which case you specify the computer name and its implied configurations. The configuration definition contains specific information concerning memory size. The computer name and the MEMORY SIZE clause are used for documentary purposes only.
2. If the PROGRAM COLLATING SEQUENCE clause is not specified, the NATIVE collating sequence is used. Refer to the *User's Guide*.
3. If the PROGRAM COLLATING SEQUENCE clause is specified, the program collating sequence is the collating sequence associated with the alphabet name specified in that clause.

-
4. The program collating sequence established in the OBJECT-COMPUTER paragraph is used to determine the truth value of nonnumeric comparisons such as those:
 - a. Explicitly specified in relation conditions (refer to "Relation Condition" on page 7-9)
 - b. Explicitly specified in condition-name conditions. Refer to "Condition-Name Condition (Conditional Variable)" on page 7-14.
 5. The PROGRAM COLLATING SEQUENCE clause is also applied to any nonnumeric merge or sort keys unless the COLLATING SEQUENCE phrase of the respective SORT or MERGE statement is specified. Refer to "SORT Statement" on page 13-21.
 6. The PROGRAM COLLATING SEQUENCE clause applies only to the program in which it is specified.
 7. The SEGMENT-LIMIT clause is documentary only. Refer to "SEGMENT-LIMIT" on page 16-7.

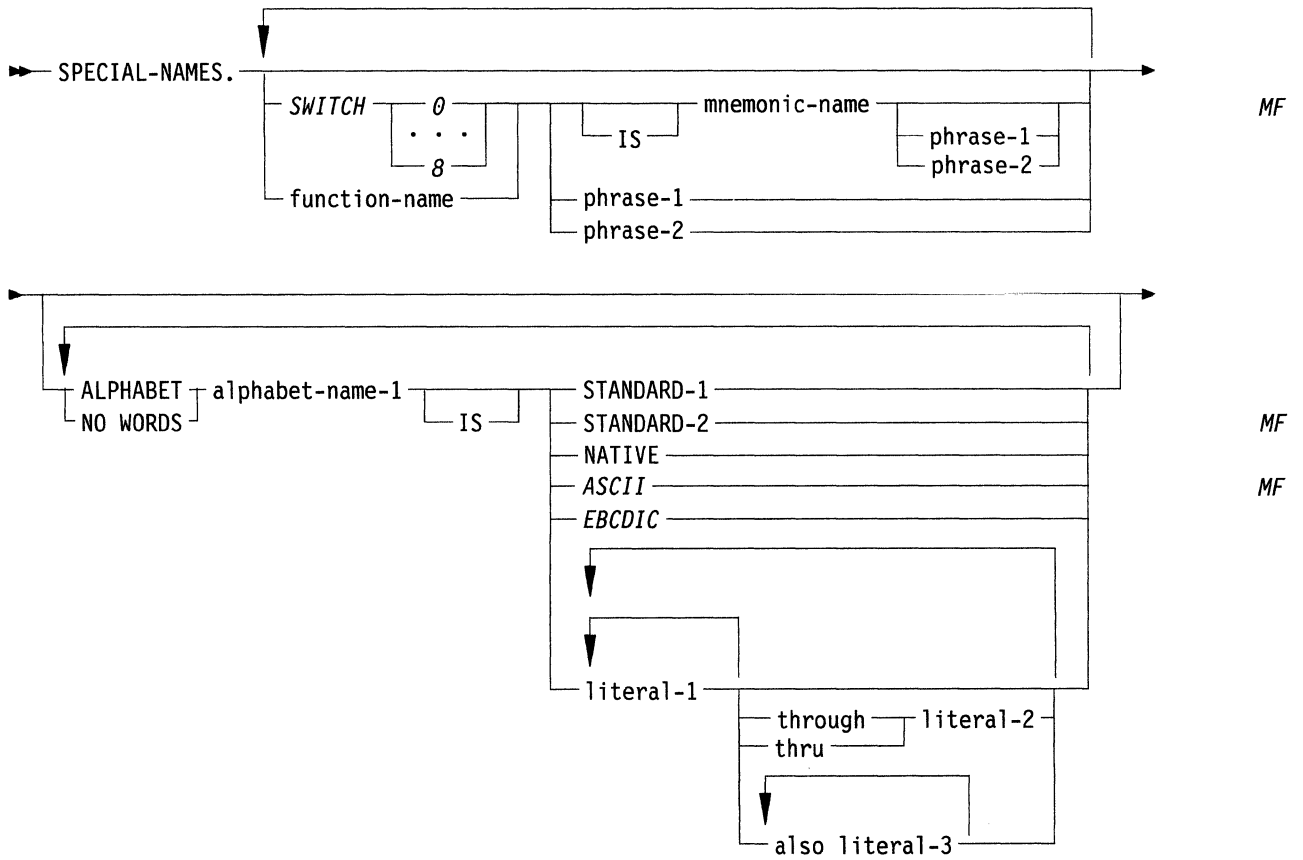
SPECIAL-NAMES Paragraph

Function

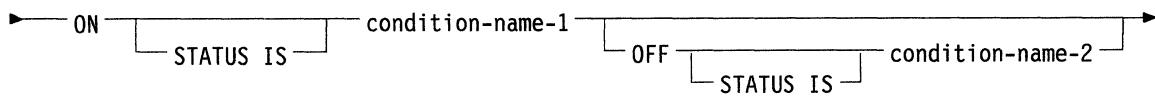
The SPECIAL-NAMES paragraph provides a means for specifying the currency sign, choosing the decimal point, specifying symbolic character, relating implementer names to user-specified mnemonic names, relating alphabet names to character sets or collating sequences, and relating class names to sets of characters.

General Format

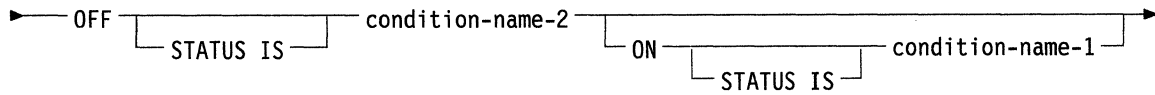
The following figure shows the general format of the SPECIAL-NAMES paragraph:

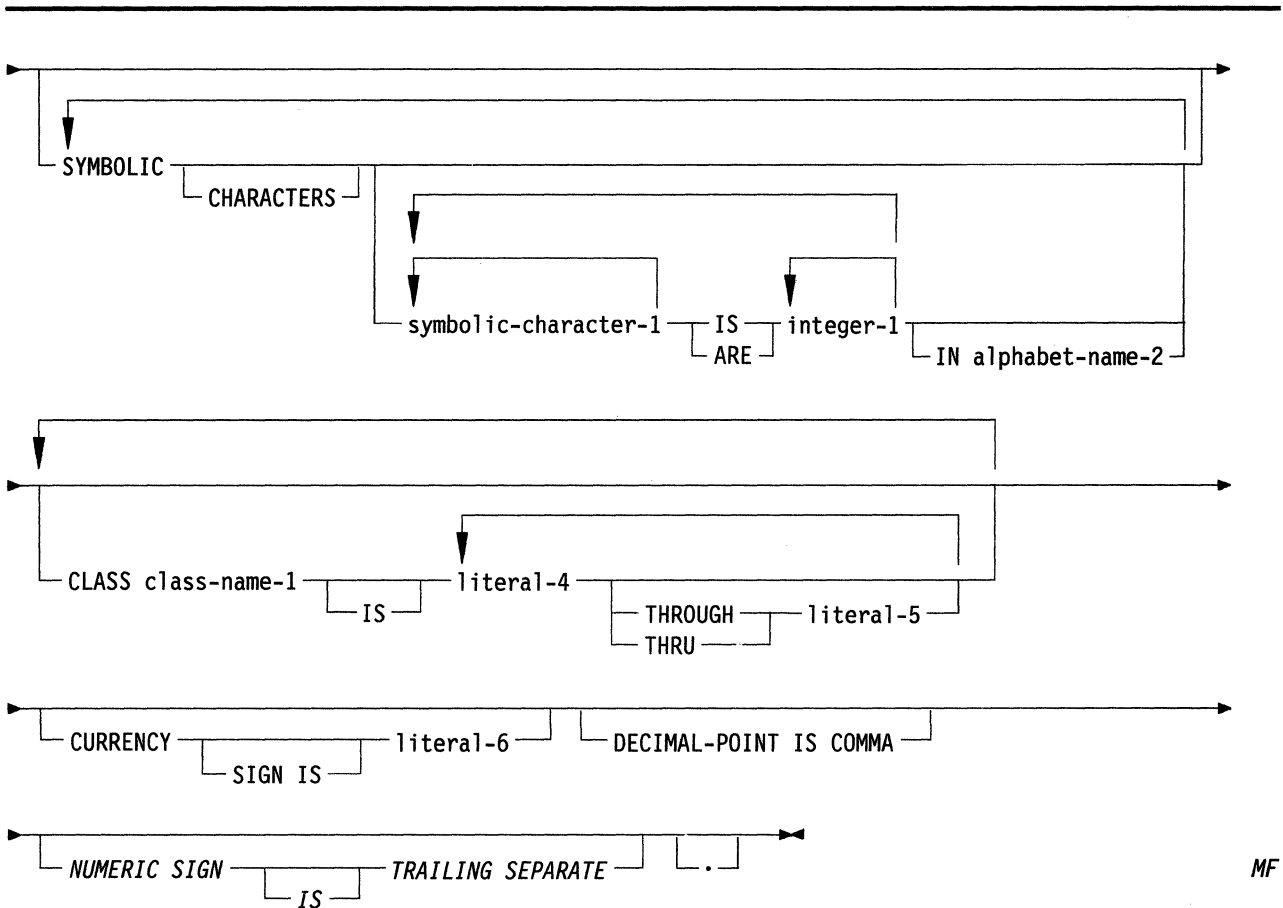


where phrase-1 is:



phrase-2 is:





Note: The optional separator period at the end of the format must be used if any of the optional clauses are selected.

Syntax Rules

The following syntax rules apply to the SPECIAL-NAMES paragraph:

1. Mnemonic names can be any COBOL user-defined word and at least one constituent character must be alphabetic.
2. *function-name* refers to a system device or function used by the AIX VS COBOL system. *function-name* refers to an external switch if, and only if, its name is one of UPSI-0 through UPSI-8. OSVS
3. If *function-name* refers to an external switch or the SWITCH option is used, the associated mnemonic name may not be specified anywhere except in the SET statement. It is recommended that at least one condition name be associated with it, but no condition name is required. MF
4. If *function-name* does not refer to an external switch, the associated mnemonic name may be specified only in the ACCEPT, DISPLAY, SEND, or WRITE statements. A condition name cannot be associated with such an implementer name.
5. If the literals specified in the literal phrase of the alphabet-name-1 clause are numeric, each must be an unsigned integer and have a value within the range of one through the maximum number of characters in the native character set.
6. If the literals specified in the literal phrase of the alphabet-name-1 clause are nonnumeric and associated with a THROUGH or ALSO phrase, each must be one character in length.

-
7. If the literal phrase of the alphabet-name-1 clause is specified, do not specify a given character more than once in an alphabet-name clause.
 8. The words THRU and THROUGH are equivalent.
 9. *The words STANDARD-1 and ASCII are synonymous.* *MF*
 10. The reserved word IS is never required in the SPECIAL-NAMES paragraph.
 11. If the literals specified in the literal-4 phrase are numeric, each must be unsigned integers and have a value within the range of one through the maximum number of characters in the native character set.
 12. If the literals specified in the literal-4 phrase are nonnumeric and associated with a THROUGH phrase, each must be one character in length.
 13. literal-1, ... literal-5 must not specify a symbolic character figurative constant.
 14. The same symbolic-character-1 must appear only once in a SYMBOLIC CHARACTERS clause.
 15. The relationship between each symbolic-character-1 and the corresponding integer-1 is determined by position in the SYMBOLIC CHARACTERS clause. The first symbolic-character-1 is paired with the first integer-1, the second symbolic-character-1 is paired with the second integer-1, and so on.
 16. There must be a one-to-one correspondence between occurrences of symbolic-character-1 and occurrences of integer-1.
 17. literal-6 must not be a figurative constant.
 18. Items defined in the SPECIAL-NAMES paragraph do not have the GLOBAL attribute.

DBCS Support

19. literal-6 must not be the uppercase alphabetic character G, for programs that use Double-Byte Character Set (DBCS) data items.

End of DBCS Support

General Rules

The following general rules apply to the SPECIAL-NAMES paragraph:

1. External switches are set at run time by the operator. The setting may be determined in the program by testing the associated condition names.
2. If mnemonic name is associated with an external switch, the status of that switch may be altered by execution of a Format 1 SET statement. Refer to "SET Statement" on page 7-84.
3. The alphabet-name-1 clause provides a means for relating a name to a specified character code set and/or collating sequence. When alphabet-name-1 is referenced in the PROGRAM COLLATING SEQUENCE clause (refer to "OBJECT-COMPUTER Paragraph" on page 5-6) or the COLLATING SEQUENCE phrase of a SORT or MERGE statement (refer to "SORT Statement" on page 13-21 and "MERGE Statement" on page 13-14), the alphabet-name-1 clause specifies a collating sequence. When alphabet-name-1 is referenced in a CODE-SET clause in a file description entry (refer to "File Description — Complete Entry Skeleton" on page 8-35), the alphabet-name clause specifies a character code set.

-
- a. If the STANDARD-1, or ASCII phrase is specified, the character code set or collating sequence identified is the American Standard Code for Information Interchange, as defined in American National Standard X3.4-1968. If the STANDARD-2 phrase is specified, the character code set identified is the International Reference Version of the ISO 7-bit code, as defined in International Standard 646, 7-bit Coded Character Set for Information Processing Interchange. MF
- b. If the NATIVE phrase is specified, the native character code set or native collating sequence is used. The native code set is ASCII, as defined in ANSI publication X3.4-1968. The native collating sequence is either ASCII or EBCDIC, as specified by the AIX VS COBOL system directive. Refer to the *User's Guide* for details of the ASCII code set and the ASCII and EBCDIC collating sequences and their correspondence.
- c. *If the EBCDIC phrase is specified, the character code set or collating sequence identified is EBCDIC.* MF
- d. If the literal phrase is specified, the alphabet-name-1 may not be referenced in a CODE-SET clause. Refer to "CODE-SET Clause" on page 8-40. The collating sequence identified is that defined according to the following rules:
- 1) The value of each literal specifies:
 - a) The ordinal number of a character within the native character set, if the literal is numeric. This value must not exceed the value which represents the number of characters in the native character set.
 - b) The actual character within the native character set, if the literal is nonnumeric. If the value of the nonnumeric literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.
 - 2) The order in which the literals appear in the ALPHABET clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified.
 - 3) Any characters within the native collating sequence, which are not explicitly specified in the literal phrase, assume a position, in the collating sequence being specified, greater than any of the explicitly specified characters. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.
 - 4) If the THROUGH phrase is specified, the set of contiguous characters in the native character set beginning with the character specified by the value of literal-1, and ending with the character specified by the value of literal-2, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.
 - 5) If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1 and literal-3 are assigned to the same position in the collating sequence being specified, or in the character code set that is used to represent the data.
4. The character that has the highest ordinal position in the program collating sequence specified is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position in the program collating sequence, the last character specified is associated with the figurative constant HIGH-VALUE.
 5. The character that has the lowest ordinal position in the program collating sequence specified is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position in the program collating sequence, the first character specified is associated with the figurative constant LOW-VALUE.

6. literal-6 that appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character and must not be one of the following characters.

- Digits 0 through 9
- Uppercase alphabetic characters A, B, C, D, P, R, S, V, X, Z, or the space; lowercase alphabetic characters a through z, (*except e, f, g, h, i, j, k, m, n, o, q, t, u, w, y*) *MF*
- Special characters * + - , . ; () " / or =.

If the literal clause is not present, only the currency sign defined in the COBOL character set is used in the PICTURE clause.

7. The DECIMAL-POINT IS COMMA clause means that the function of comma and period are exchanged in the character string of the PICTURE clause and in numeric literals.

8. *If the NUMERIC SIGN clause is specified, the default for signed numeric items is for the sign to be stored as a trailing separate character.* *MF*

9. *If function-name does not reference an external switch, it may be selected from Table 5-1 on page 5-14.*

10. The CLASS clause provides a means for relating a name to the specified set of characters listed in that clause. class-name-1 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 which constitute class-name-1.

The value of each literal specifies:

- a. The ordinal number of a character within the native character set (if the literal is numeric). This value must not exceed the value that represents the number of characters in the native character set.
- b. The actual character within the native character set (if the literal is nonnumeric). 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-1.
- c. If the THROUGH phrase is specified, the contiguous characters in the native character set (beginning with the character specified by the value of literal-4 and ending with the character specified by the value of literal-5) are included in the set of characters identified by class-name-1. In addition, the contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in ascending or descending sequence.

Table 5-1. Function-Name Reference		
Function-name	Meaning	Used in
<i>C01 through C12</i>	<i>Skip to channel 1 through 12, respectively</i>	<i>WRITE ADVANCING statement OSVS VSC2</i>
<i>CONSOLE</i>	<i>Console device</i>	<i>ACCEPT and DISPLAY statements</i>
<i>CSP</i>	<i>Suppress space</i>	<i>WRITE ADVANCING statement</i>
<i>S01, S02</i>	<i>Bin select 1 or 2 on card-punch devices statement</i>	<i>WRITE ADVANCING</i>
<i>SYSIN SYSIPT</i>	<i>System input</i>	<i>ACCEPT statement</i>
<i>SYSOUT SYSLIST SYSLST SYSPUNCH SYSPNCH</i>	<i>System output</i>	<i>DISPLAY statement</i>
<i>COMMAND-LINE</i>	<i>Command transfer</i>	<i>ACCEPT and DISPLAY statements MF</i>
<i>TAB</i>	<i>Skip to Vertical Tabulation Position (inserts ASCII X'0B' in the output record as appropriate)</i>	<i>WRITE ADVANCING statement</i>
<i>FORMFEED</i>	<i>Skip to a new page (inserts ASCII X'0C' in the output record as appropriate)</i>	<i>WRITE ADVANCING statement</i>

Example

The following example shows the object computer and special names for the SPECIAL-NAMES paragraph:

```

OBJECT-COMPUTER.
  IBM-RISC-6000
  MEMORY SIZE 4000000 characters
  PROGRAM COLLATING SEQUENCE IS INFO-CODE.
SPECIAL-NAMES.
  SWITCH 0 ON STATUS IS TESTRUN
  SWITCH 0 OFF STATUS IS PRODRUN
  CURRENCY SIGN IS Y
  DECIMAL-POINT IS COMMA
  ALPHABET INFO-CODE IS ASCII.

```

Chapter 6. Data Division

Contents

About This Chapter	6-5
WORKING-STORAGE SECTION	6-6
General Format	6-6
Noncontiguous Working Storage (77-Level-Description-Entry)	6-6
Working-Storage Records (Record-Description-Entry)	6-6
Record Description Structure	6-7
Initial Value	6-7
Data Description — Complete Entry Skeleton	6-7
Function	6-7
General Format	6-7
Syntax Rules	6-9
General Rules	6-9
BLANK WHEN ZERO Clause	6-11
Function	6-11
General Format	6-11
Syntax Rule	6-11
General Rules	6-11
Data-Name or FILLER Clause	6-12
Function	6-12
General Format	6-12
Syntax Rule	6-12
General Rules	6-12
JUSTIFIED Clause	6-13
Function	6-13
General Format	6-13
Syntax Rules	6-13
General Rules	6-13
Level Number	6-15
Function	6-15
General Format	6-15
Syntax Rules	6-15
General Rules	6-16
Example	6-17
PICTURE Clause	6-18
Function	6-18
General Format	6-18
Syntax Rules	6-18
General Rule	6-19
Elementary Item Size	6-20
Symbols Used	6-21
Editing Rules	6-22
Precedence Rules	6-27
REDEFINES Clause	6-29
Function	6-29
General Format	6-29
Syntax Rules	6-29
General Rules	6-30
Example	6-30
RENAMES Clause	6-32
Function	6-32
General Format	6-32
Syntax Rules	6-32
General Rules	6-33
Example	6-34
SIGN Clause	6-35
Function	6-35
General Format	6-35
Syntax Rules	6-35
General Rules	6-35

SYNCHRONIZED Clause	6-37
Function	6-37
General Format	6-37
Syntax Rules	6-37
General Rules	6-37
USAGE Clause	6-39
Function	6-39
General Format	6-39
Syntax Rules	6-39
General Rules	6-40
VALUE Clause	6-41
Function	6-41
General Format	6-41
Syntax Rules	6-41
General Rules	6-42
Condition-Name Rules	6-43
Constant-Name Rules	6-43
Data Description Entries Other Than <i>CONDITION-NAMES</i> and <i>CONSTANT-NAMES</i>	6-44



About This Chapter

This chapter describes the Data Division in the Nucleus of a COBOL source program.

The Data Division specifies all the data to be processed by the executable program. Different types of data are described by different sections:

- **WORKING-STORAGE SECTION** - For internal data
- **FILE SECTION FD** - For externally stored data
- **FILE SECTION SD** - For sort-merge files
- **LINKAGE SECTION** - For inter-program communication arguments
- **COMMUNICATION SECTION** - For messages between program and local or remote communication devices
- **SCREEN SECTION** - For information flow between program and console.

The **WORKING-STORAGE SECTION** is discussed in this chapter. Other sections will be covered in later chapters.

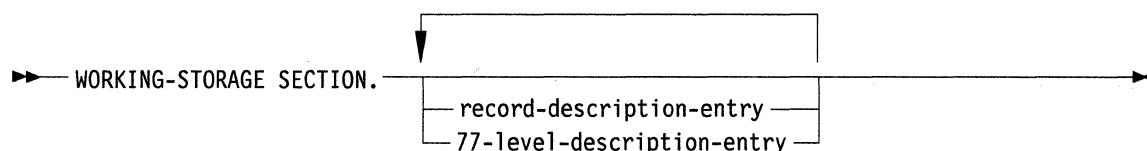
WORKING-STORAGE SECTION

The WORKING-STORAGE SECTION is composed of the section header, followed by data description entries for noncontiguous data items and/or record description entries. Each WORKING-STORAGE SECTION record name and noncontiguous item name should be unique since it cannot be qualified. Subordinate data-names need not be unique if they can be made unique by qualification.

If no reference is made to a given data-name or record name, it is not OSVS VSC2 required that the data or record name should be unique by qualification.

General Format

The following figure shows the general format for the WORKING-STORAGE SECTION:



Noncontiguous Working Storage (77-Level-Description-Entry)

Items and constants in the WORKING-STORAGE SECTION that bear no hierarchical relationship to one another need not be grouped into records, if they do not need to be further subdivided. Instead, they are classified as noncontiguous elementary items and are defined in a separate data description entry that begins with the special level number 77.

The following data clauses are required in each data description entry:

- Level number 77
- Data-name
- The PICTURE clause, the USAGE IS INDEX clause, or the USAGE IS *VSC2* POINTER clause.

Other data description clauses are optional and can be used to complete the description of the item.

Working-Storage Records (Record-Description-Entry)

Data elements in the WORKING-STORAGE SECTION that bear a definite hierarchical relationship to one another must be grouped into records according to the rules for formation of record descriptions. All clauses that are used in record descriptions in the FILE SECTION can be used in record descriptions in the WORKING-STORAGE SECTION.

Record Description Structure

A record description consists of a set of data description entries that describe the characteristics of a particular record. Each data description entry consists of a level number followed by the data-name or FILLER clause, if specified, followed by a series of independent clauses as required. A record description may have a hierarchical structure. Therefore, the clauses used in an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description and the elements allowed in a record description entry are explained in "Concept Of Levels" on page 2-16 and in "Data Description — Complete Entry Skeleton" in this section.

Initial Value

The initial value of an item in the WORKING-STORAGE SECTION, except an index data item, may be specified by using the VALUE clause with the data item. The item value of an index data item or any data item not associated with a VALUE clause is undefined.

Data Description — Complete Entry Skeleton

This section describes the function, general format, syntax and general rules for the Data Description — complete entry skeleton.

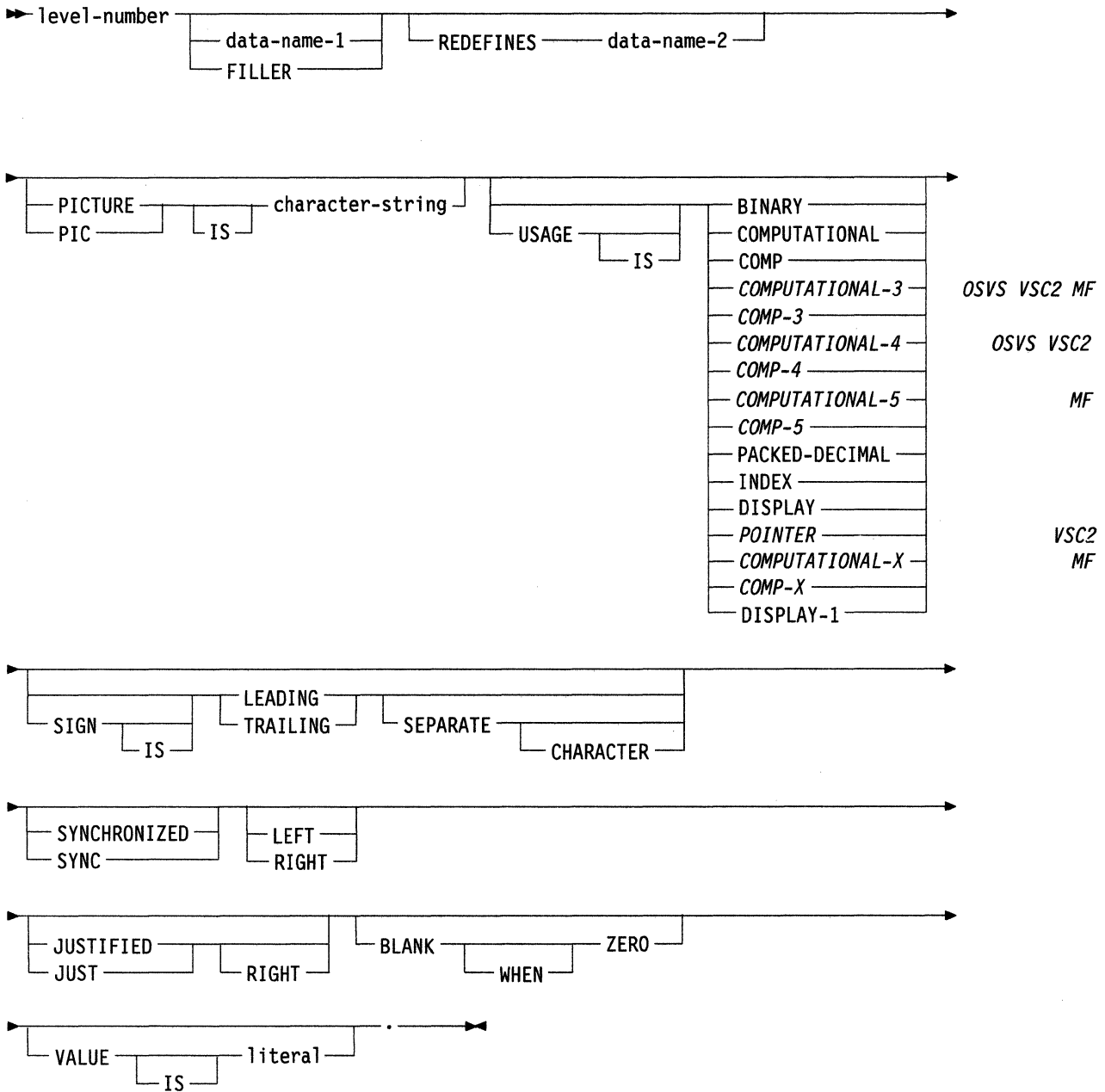
Function

A data description entry specifies the characteristics of a particular item of data.

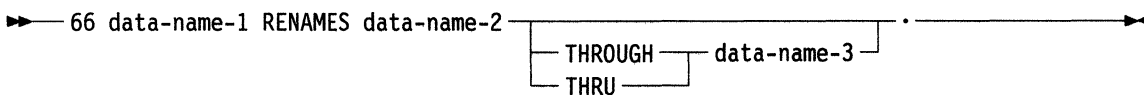
General Format

The following figure shows the formats of the data description complete entry skeleton:

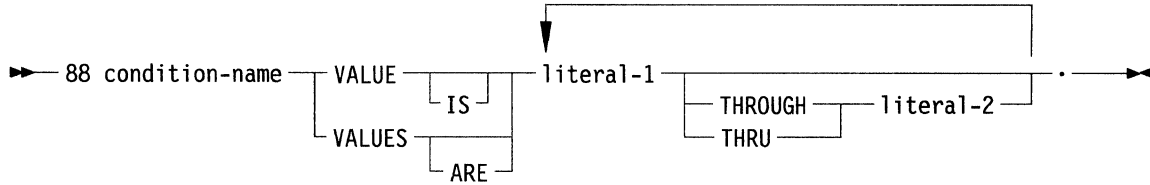
Format 1



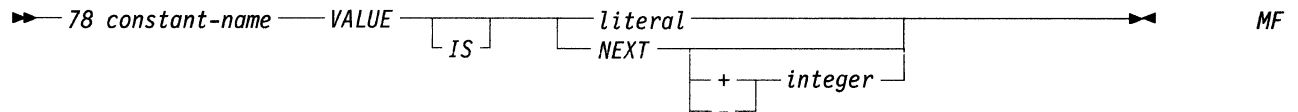
Format 2



Format 3



Format 4



Syntax Rules

The syntax rules for the data description are as follows:

1. The level-number in Format 1 may be any number from 01 through 49 or 77.
2. The clauses may be written in any order with the two following exceptions. The data-name-1 or FILLER clause must immediately follow the level-number. The REDEFINES clause must immediately follow the data-name-1 clause or FILLER clause. Otherwise, the REDEFINES clause must immediately follow the level-number.
3. The PICTURE clause must be specified for all elementary items other than an index data item. Specification of the PICTURE clause is prohibited with an index data item.
4. The words THRU and THROUGH are equivalent.

General Rules

The following general rules apply to the data description:

1. The clauses SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO, must not be specified except for an elementary data item. *The SYNCHRONIZED clause may be specified for a group item.* OSVS
2. Format 3 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. Format 3 contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must immediately follow the entry describing the item with which the condition-name is associated. A condition-name cannot be associated with any of the following data description entries that contain a level-number:
 - a. Another condition-name
 - b. A level 77 item
 - c. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY)
 - d. An index data item (refer to "USAGE IS INDEX Clause" on page 12-12)
 - e. *A constant-name.* MF

-
3. *Format 4 defines a constant-name that is a symbolic name representing a constant value assigned to it when the source code is passed through IBM AIX VS COBOL system. The AIX VS COBOL system replaces each reference to a constant-name by its value.*

MF

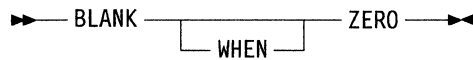
BLANK WHEN ZERO Clause

Function

The BLANK WHEN ZERO clause permits the blanking of an item when its value is zero.

General Format

The following figure shows the general format of the BLANK WHEN ZERO clause:



Syntax Rule

The following syntax rule applies to the BLANK WHEN ZERO clause:

1. The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric (with implicit or explicit USAGE IS DISPLAY) or numeric-edited. Refer to "PICTURE Clause" on page 6-18.

General Rules

The following general rules apply to the BLANK WHEN ZERO clause:

1. When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero.
2. When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric-edited.
3. *If the BLANK WHEN ZERO clause is specified for an elementary item OSVS VSC2 whose PICTURE clause includes a zero suppression symbol Z or *, zero suppression overrides BLANK WHEN ZERO.*

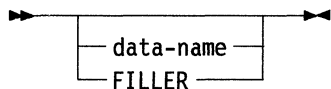
Data-Name or FILLER Clause

Function

The data-name specifies the name of the data being described. The word **FILLER** may be used to specify an elementary item of the logical record that cannot be referred to explicitly.

General Format

The following figure shows the general format for the data-name or **FILLER** clause:



Syntax Rule

In the **FILE**, **WORKING-STORAGE**, **COMMUNICATION**, and **LINKAGE SECTIONS**, a data-name or the key word **FILLER**, if either is specified, must be the first word following the level number in each data description entry.

General Rules

The following general rules apply to the data-name or **FILLER** clause:

1. The key word **FILLER** may be used to name an elementary item or group in a record. Under no circumstances can a **FILLER** item be referred to explicitly. However, the key word **FILLER** may be used as a conditional variable because such use does not require explicit reference to the **FILLER** item but only to the value contained therein.
2. If the data-name or **FILLER** clause is omitted, the data item being described is treated as though **FILLER** had been specified.

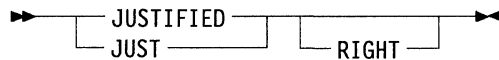
JUSTIFIED Clause

Function

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data item.

General Format

The following figure shows the general format for the JUSTIFIED clause:



Syntax Rules

The following syntax rules apply to the JUSTIFIED clause:

1. The JUSTIFIED clause can be specified only at the elementary item level.
2. JUST is an abbreviation for JUSTIFIED.
3. The JUSTIFIED clause cannot be specified for a data item described as numeric or for which editing is specified.
4. The JUSTIFIED clause cannot be specified for an index (refer to “USAGE IS INDEX Clause” on page 12-12) or a pointer data item. VSC2

General Rules

The following general rules apply to the JUSTIFIED clause:

1. When a receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the rightmost character position in the data item with space fill for the leftmost character positions.

The contents of the sending data items are not taken into account (trailing spaces within the sending data item are not suppressed).

For example, if a data item PIC X(4) whose value is ‘A_ _ _’ (that is, A followed by three spaces), is moved into a data item PIC X(6) JUSTIFIED the result will be ‘_ _A_ _’. If the same data item is moved to one with PIC X(3) JUSTIFIED, the result will be ‘_ _ _’ (that is, the leftmost character is truncated).

2. When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply. Refer to “Standard Alignment Rules” on page 2-19.

DBCS Support

3. The **JUSTIFIED RIGHT** clause may be specified for a DBCS item. When **JUSTIFIED RIGHT** 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.

End of DBCS Support

Level Number

Function

The level number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for working storage items, linkage items, condition names, *constant names*, and the RENAMEs clause.

MF

General Format

The following figure shows the general format for the level number:

▶▶—level-number—▶▶

Syntax Rules

The following syntax rules apply to the level number:

1. A level number is required as the first element in each data description entry.
2. Data description entries subordinate to an FD, CD, or SD entry must have level numbers with the values 01 through 49, 66, 78, or 88. *MF*
Refer to “File Description — Complete Entry Skeleton” on page 8-35.
3. Data description entries in the REPORT and SCREEN SECTIONS must have level numbers with the values 01 through 49, or 78. *MF*
4. Data description entries in the WORKING-STORAGE and LINKAGE SECTIONS must have level numbers with the values 01 through 49, 66, 77, 78, or 88. *MF*
5. A level number may be a one- or two-digit number.

General Rules

The following general rules apply to the level number:

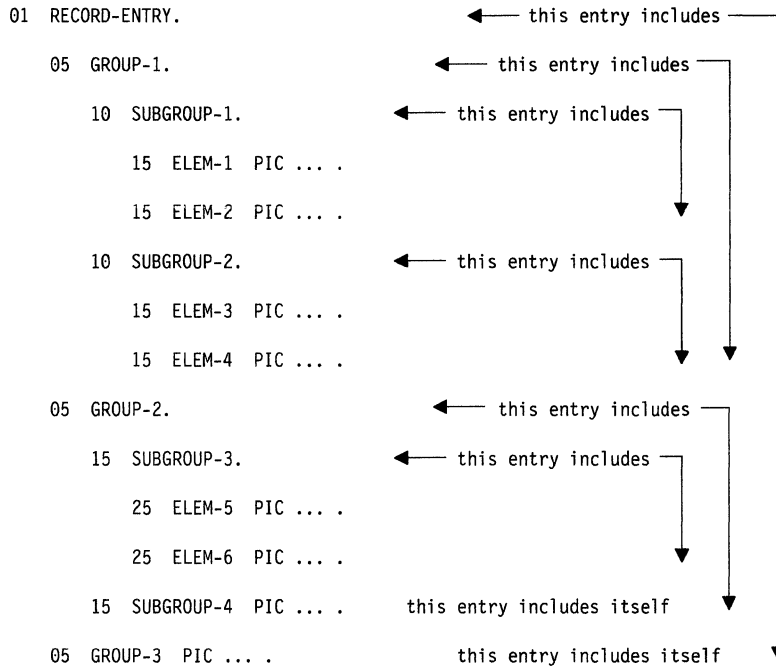
1. The level number 01 identifies the first entry in each record description.
2. The following special level numbers have been assigned to entries where there is no concept of level hierarchy:
 - a. Level number 77 is assigned to identify noncontiguous working-storage data items, and noncontiguous linkage data items. It can be used only as described by Format 1 of the data description skeleton.
 - b. Level number 66 is assigned to identify RENAMEs entries. It can be used only as described in Format 2 of the data description skeleton.
 - c. Level number 88 is assigned to entries that define condition-names associated with a conditional variable. It can be used only as described in Format 3 of the data description skeleton.
 - d. *Level number 78 is assigned to entries that define constant-names. It can be used only as described in Format 4 of the data description skeleton.* *MF*
3. Multiple level 01 entries subordinate to a CD, FD, or SD entry represent implicit redefinitions of the same area.

Example

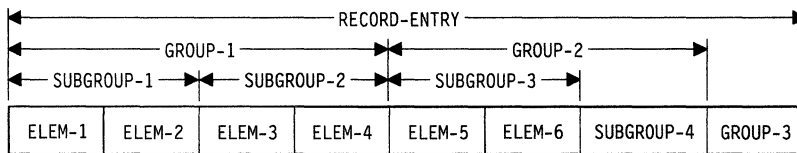
The following is an example of level-number concepts:

The COBOL record-description entry written as follows:

Is subdivided as indicated below:



Its storage arrangement is illustrated below:



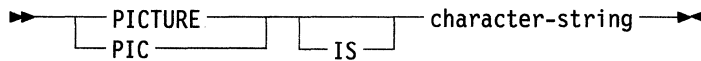
PICTURE Clause

Function

The PICTURE clause describes the general characteristics and editing requirements of an elementary item.

General Format

The following figure shows the general format for the PICTURE clause:



Syntax Rules

The following syntax rules apply to the PICTURE clause:

1. A PICTURE clause can be specified only at the elementary item level.
2. A character-string consists of certain allowable combinations of characters in the COBOL character set. The characters are used as symbols. The allowable combinations determine the category of the elementary item.
3. The maximum number of characters allowed in the character string is 30.
4. The PICTURE clause must be specified for all elementary items except an index data item or the subject of a RENAMES clause. Other use of this clause is prohibited.
5. PIC is an abbreviation for PICTURE.
6. The asterisk when used as the zero suppression symbol and the clause BLANK WHEN ZERO should not appear in the same entry. *However, OSVS this construct is permitted.*
7. An integer in parentheses following a character indicates that the character is repeated the number of times shown by the integer.

General Rule

The following seven categories of data can be described with a PICTURE clause:

- Alphabetic
- Numeric
- Alphanumeric
- Alphanumeric-edited
- Numeric-edited

DBCS Support

- DBCS
- DBCS-edited.

End of DBCS Support

General rules within these categories follow.

Alphabetic Data Rules

The following alphabetic data rules apply to the PICTURE clause:

1. The PICTURE character string can only contain the symbols A and B.
2. Its contents when represented in standard data format must be one or more alphabetic characters.

Numeric Data Rules

The following numeric data rules apply to the PICTURE clause:

1. The PICTURE character string can only contain the symbols 9, P, S, and V. The number of digit positions that can be described by the PICTURE character string must range from 1 to 18 inclusive.
2. If unsigned, the data in standard data format must be a combination of the Arabic numerals 0 through 9. If signed, the item may also contain a +, -, or other representation of an operational sign. Refer to "SIGN Clause" on page 6-35.

Numeric data may also be held in formats other than standard data formats. Refer to "USAGE Clause" on page 6-39, and "Selection of Character Representation and Radix" on page 2-20.

Alphanumeric Data Rules

The following alphanumeric data rules apply to the PICTURE clause:

1. The PICTURE character string is restricted to certain combinations of the symbols A, X, 9, and the item is treated as if the character string contained all Xs. A PICTURE character string that contains all A's or all 9s does not define an alphanumeric item.
2. The contents, when represented in standard data format, may consist of any characters in the character set.

Alphanumeric-Edited Data Rules

The following alphanumeric-edited data rules apply to the PICTURE clause:

1. The PICTURE character string is restricted to certain combinations of the symbols A, X, 9, B, 0, and / as follows:
 - a. A character string must contain at least one B and one X, one 0 (zero) and one X, or at least one / (stroke) and one X, or

-
- b. The character string must contain at least one 0 (zero) and one A or at least one / (stroke) and one A.
 2. The contents when represented in standard data format are allowable characters in the set of the computer.

Numeric-Edited Data Rules

The following numeric-edited data rules apply to the PICTURE clause:

1. The PICTURE character string is restricted to certain combinations of the symbols B / P V Z 0 9 , . * + - CR DB and the currency symbol. The allowable combinations are determined from the order of precedence of symbols and the editing rules as follows:
 - a. The number of digit positions that can be represented in the PICTURE character string must range from 1 to 18 inclusive.
 - b. The character string must contain at least one 0 B / Z * + , . - CR DB or the currency symbol.
 - c. The maximum storage represented by the PICTURE character string is 512 bytes.
2. The contents of the character positions of these symbols that are allowed to represent a digit in standard data format must be one of the numerals.

DBCS Support

DBCS Data Rules

The following DBCS data rules apply to the PICTURE clause:

1. The PICTURE character-string can contain only the symbol G.
2. Each G represents a single DBCS character position (2 bytes).
3. USAGE DISPLAY-1 must be specified.
4. Any associated VALUE clause must specify a DBCS literal or the figurative constant SPACE/SPACES.

DBCS-Edited Data Rules

The following DBCS-edited data rules apply to the PICTURE clause:

1. The PICTURE character-string can contain the symbols G and B.
2. Each G represents a single DBCS character position (2 bytes).
3. USAGE DISPLAY-1 must be specified.
4. Any associated VALUE clause must specify a DBCS literal or the figurative constant SPACE/SPACES.

End of DBCS Support

Elementary Item Size

The size of an elementary item, where size means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols that represent character positions. An integer that is enclosed in parentheses following the symbols A X 9 P Z , * B / 0 + - or the currency symbol indicates the number of consecutive occurrences of the symbol. The S V . CR and DB symbols may appear only once in a given PICTURE.

Symbols Used

The functions of the symbols used to describe an elementary item are as follows:

- A** Each A in the character string represents a character position that can contain only a letter of the alphabet or a space.
- B** Each B in the character string represents a character position into which the space character will be inserted.

DBCS Support

For a DBCS item, each picture symbol B represents a single DBCS character position containing a DBCS space.

Each B counts as one character.

- G** Each G in the character string represents a DBCS character position, occupying 2 bytes of storage. The PICTURE symbol G cannot be specified for a non-DBCS item.

Each G counts as one character.

End of DBCS Support

- P** Each P indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is 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 numeric items.

The scaling position character P can appear only to the left or right as a continuous string of Ps within a PICTURE description. The scaling position character P implies an assumed decimal point (to the left of Ps if Ps are leftmost PICTURE characters and to the right if 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.

The character P and the insertion character . (period) cannot both occur in the same PICTURE character string. If, in any operation involving conversion of data from one form of internal representation to another, the data item being converted is described with the PICTURE character P, each digit position described by a P is considered to contain the value zero. The size of the data item is considered to include the digit positions so described.

- S** The letter S is used in a character string to indicate the presence (but neither the representation nor, necessarily, the position) of an operational sign and it must be written as the leftmost character in the PICTURE. The S is not counted in determining the size (in terms of standard data format characters) of the elementary item unless the entry is subject to a SIGN clause that specifies the optional SEPARATE CHARACTER phrase. Refer to "SIGN Clause" on page 6-35.

- V** The V is used in a character string to indicate the location of the assumed decimal point and may only appear once in a character string. The V does not represent a character position and therefore is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string the V is redundant.

- X** Each X in the character string is used to represent a character position that contains any allowable character from the character set.

- Z** Each Z in a character string may only be used to represent the leftmost numeric character positions that will be replaced by a space character when the contents of that character position is zero. Each Z is counted in the size of the item.

-
- 9** Each 9 in the character string represents a character position that contains a numeral and is counted in the size of the item.
- 0** Each 0 (zero) in the character string represents a character position into which the numeral 0 will be inserted. The 0 is counted in the size of the item.
- /** Each / (stroke) in the character string represents a character position into which the stroke character will be inserted. The / is counted in the size of the item.
- ,** Each , (comma) in the character string represents a character position into which the character , will be inserted. This character position is counted in the size of the item. The insertion character , must not be the last character in the PICTURE character string.
- .** When the character . (period) appears in the character string, it is an editing symbol that represents the decimal point for alignment purposes. It also represents a character position into which the character . will be inserted. The character . is counted in the size of the item. For a given program, the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in the PICTURE clause. The insertion character , must not be the last character in the PICTURE character string.
- +, -, DB, CR** These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character string and each character used in the symbol is counted in determining the size of the data item.
- *** Each * (asterisk) in the character string represents a leading numeric character position into which an asterisk will be placed when the contents of that position is zero. Each * is counted in the size of the item.
- \$** The currency symbol in the character string represents a character position into which a currency symbol is to be placed. The currency symbol in a character string is represented by either the currency sign or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item.

Editing Rules

There are two general methods of performing editing in the PICTURE clause. Editing can be done either by insertion or by suppression and replacement. There are four types of insertion editing available:

- Simple insertion
- Special insertion
- Fixed insertion
- Floating insertion.

There are two types of suppression and replacement editing:

- Zero suppression and replacement with spaces
- Zero suppression and replacement with asterisks.

The type of editing that may be performed on an item is dependent upon the category to which the item belongs. Table 6-1 on page 6-23 specifies the type of editing that may be performed on a given category.

Category	Type of Editing
Alphabetic	Simple insertion B only
Numeric	None
Alphanumeric	None
Alphanumeric-Edited	Simple insertion 0, B, and /
Numeric-Edited	All (see note)
DBCS	None
DBCS-edited	Simple insertion

Note: Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause. Only one type of replacement may be used with zero suppression in a PICTURE clause.

Simple Insertion Editing

Simple insertion editing is valid for alphabetic, alphanumeric-edited, and numeric-edited items. The , (comma), B (space), 0 (zero), and / (stroke) are used as the insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted.

DBCS Support

This type of editing is valid for DBCS-edited items.

End of DBCS Support

Category	Valid Insertion Symbols
Alphabetic	B
Alphanumeric-edited	B 0 /
Numeric-edited	B 0 / ,

DBCS Support

DBCS-edited B

End of DBCS Support

The following example shows simple insertion editing.

PICTURE	Value of Data	Edited Result
X(10)/XX	ALPHANUMER01	ALPHANUMER/01
X(5)BX(7)	ALPHANUMERIC	ALPHA NUMERIC
A(5)BA(5)	ALPHABETIC	ALPHA BETIC
99,B999,B000	1234	01, 234, 000
99,999	12345	12,345

DBCS Support

For DBCS-edited 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.

End of DBCS Support

Special Insertion Editing

Special insertion editing is valid only for numeric-edited items. The . (period) is used as the insertion character and also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item.

The use of the assumed decimal point, represented by the symbol V and the actual decimal point, represented by the insertion character, in the same PICTURE character string is disallowed.

The following example shows special insertion editing:

PICTURE	Value of Data	Edited Results
999.99	1.234	001.23
999.99	12.34	012.34
999.99	123.45	123.45
999.99	1234.5	234.50

Fixed Insertion Editing

Fixed insertion editing is valid only for numeric-edited items. The currency symbol and the editing sign control symbols +, -, CR, DB are the insertion characters.

Consider the following conditions about fixed insertion editing:

- In fixed insertion editing, only one currency symbol and one editing sign control symbol can be specified in a PICTURE character string.
- Unless it is preceded by a + or - symbol, the currency symbol (\$) must be the leftmost character position in the character string.
- When either + or - is used as a symbol, it must represent either the leftmost or rightmost character position in the character string.
- When CR or DB is used as a symbol, it must represent the two rightmost character positions in the character string.
- Editing sign control symbols produce results depending on the value of the data item, as shown in Table 6-2.

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

The following example shows fixed insertion editing:

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
\$9999.99CR	+123.45	\$0123.45
\$9999.99DB	-123.45	\$0123.45DB

Floating Insertion Editing

Floating insertion editing is valid only for numeric-edited items. The currency symbol and editing sign control symbols + or - are the floating insertion characters and as such are mutually exclusive in a given PICTURE character string.

Floating insertion editing is indicated in a PICTURE character string by using a string of at least two of the floating insertion characters. This string of floating insertion characters may contain any of the fixed insertion symbols or have fixed insertion characters immediately to the right of this string. These simple insertion characters are part of the floating string.

The leftmost character of the floating insertion string represents the leftmost limit of the floating symbol in the data item. The rightmost character of the floating string represents the rightmost limit of the floating symbols in the data item.

The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. Nonzero numeric data may replace all the characters at or to the right of this limit.

In a PICTURE character string, there are only two ways of representing floating insertion editing:

- Any or all of the leading numeric character positions on the left of the decimal point are represented by the insertion character.
- All of the numeric character positions in the PICTURE character string are represented by the insertion character.

If the insertion characters are only to the left of the decimal point in the PICTURE character string, a single floating insertion character will be placed in the character position immediately preceding either the decimal point or the first nonzero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character string. The character positions preceding the insertion character are replaced with spaces.

If all numeric character positions in the PICTURE character string are represented by the insertion character, the result depends on the value of the data. If the value is zero, the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character string for the receiving data item must be the number of characters in the sending data item, plus the number of nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

The following example shows floating insertion editing:

PICTURE	Value of Data	Edited Result
\$\$\$\$.99	.123	\$.12
\$\$\$9.99	.12	\$0.12
\$\$,\$\$\$,999.99	-1234.56	\$1,234.56
++,+++,999.99	-123456.789	-123,456.78
\$\$,\$\$\$,\$\$\$\$.99CR	-1234567	\$1,234,567.00CR
++,+++,+++.+++	0000.00	

Zero Suppression Editing

The suppression of leading zeros in numeric character positions is indicated by the use of the alphabetic character Z or the character * (asterisk) as suppression symbols in a PICTURE character string. These symbols are mutually exclusive in a given PICTURE character string. Each suppression symbol is counted in determining the size of the item. If Z is used, the replacement character will be the space and if the asterisk is used, the replacement character will be *.

Zero suppression and replacement is indicated in a PICTURE character string by using a string of one or more of the allowable symbols to represent leading numeric character positions. The numeric character positions are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

In a PICTURE character string, there are only two ways of representing zero suppression. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. The other way is to represent all of the numeric character positions in the PICTURE character string by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data that corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character string are represented by suppression symbols and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is Z, the entire data item will be spaces. If the value is zero and the suppression symbol is *, the data item will be all * except for the actual decimal point.

The symbols +, -, *, Z, and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character string.

The following is an example of zero suppression editing:

PICTURE	Value of Data	Edited Result
****.**	0000.00	****.**
ZZZZ.ZZ	0000.00	
ZZZZ.99	0000.00	.00
****.99	0000.00	****.00
ZZ99.99	0000.00	00.00
Z,ZZZ.ZZ+	+123.456	123.45+
*,***.***+	-123.45	**123.45-
,*,***.***+	+12345678.9	12,345,678.90+
\$Z,ZZZ,ZZZ.ZZCR	+12345.67	\$ 12,345.67
\$B*,***,***.**BBDB	-12345.67	\$ ***12,345.67 DB

Precedence Rules

Figure 6-1 on page 6-28 shows the order of precedence when using characters as symbols in a character string. An X at an intersection indicates that the symbol(s) at the top of the column may precede, in a given character string, the symbol(s) at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol cs.

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.

DBCS Support

The symbol G may appear alone in a PICTURE character string. The symbol G may appear more than once in a PICTURE character string.

End of DBCS Support

In Figure 6-1 on page 6-28, nonfloating insertion symbols + and -, floating insertion symbols Z, *, +, -, and cs, and other symbol P appear twice in the PICTURE character precedence chart. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of symbol in the row and column represents its use to the right of the decimal point position.

First Symbol	Non-Floating Insertion Symbols										Floating Insertion Symbols					Other Symbols							
	B	0	/	,	.	{±}	{±}	{CR DB}	CS	{Z*}	{Z*}	{±}	{±}	CS	CS	9	{A x}	S	V	P	P	G	
Non Floating Insertion Symbols	B	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x		x		x	x	
	0	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x		x		x		
	/	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x		x		x		
	,	x	x	x	x	x	x		x	x	x	x	x	x	x	x			x		x		
	.	x	x	x	x		x		x	x		x		x		x							
	{±}																						
	{±}	x	x	x	x	x			x	x	x			x	x	x			x	x	x		
	{CR DB}	x	x	x	x	x			x	x	x			x	x	x			x	x	x		
CS						x																	
Floating Insertion Symbols	{Z*}	x	x	x	x		x		x	x													
	{Z*}	x	x	x	x	x			x	x	x								x		x		
	{±}	x	x	x	x				x			x											
	{±}	x	x	x	x	x			x			x	x						x		x		
	CS	x	x	x	x		x							x									
	CS	x	x	x	x	x	x							x	x				x		x		
Other Symbols	9	x	x	x	x	x	x		x	x		x		x		x	x	x	x		x		
	{A x}	x	x	x												x	x						
	S																						
	V	x	x	x	x		x		x	x		x		x		x		x		x			
	P	x	x	x	x		x		x	x		x		x		x		x		x			
	P						x		x									x	x		x		
G	x																					x	

Figure 6-1. PICTURE Character Precedence Chart

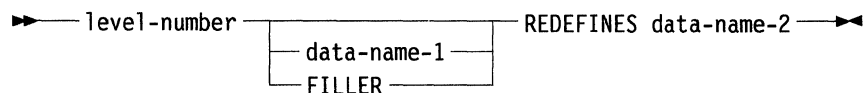
REDEFINES Clause

Function

The REDEFINES clause allows the same computer memory area to be described by different data description entries.

General Format

The following figure shows the general format of the REDEFINES clause:



Note: level-number, data-name-1, and FILLER are shown in the above format to improve clarity. level-number, data-name-1, and FILLER are not part of the REDEFINES clause.

Syntax Rules

The following syntax rules apply to the REDEFINES clause:

1. When specified, the REDEFINES clause must immediately follow data-name-1 or it may follow the PICTURE or USAGE clause. OSVS
2. The level-numbers of data-name-1 and data-name-2 must be identical but must not be 66 or 88.
3. This clause must not be used in level 01 entries in the FILE SECTION, because multiple level 01 entries subordinate to an FD or SD indicator represent implicit redefinitions of the same areas. Refer to "DATA RECORDS Clause" on page 8-42.
4. This clause must not be used in level 01 entries in the COMMUNICATION SECTION because multiple level 01 entries subordinate to a CD indicator represent implicit redefinition of the same area.
5. The data description for data-name-2 may contain a REDEFINES clause OSVS VSC2 and data-name-2 may be subordinate to an entry that contains a REDEFINES clause. Its data description cannot contain an OCCURS clause. However, data-name-2 may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause may not be subscripted or indexed. Neither the original definition nor the redefinition can include an item whose size is variable as defined in the OCCURS clause. Refer to "OCCURS Clause" on page 12-5.
6. No entry having a level-number numerically lower than the level-number of data-name-2 and data-name-1 may occur between the data description entries of data-name-2 and data-name-1.

General Rules

The following general rules apply to the REDEFINES clause.

1. Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.
2. When the level-number of data-name-1 is other than 01, it must specify the same number of character positions that the data item referenced by data-name-2 contains *except that the data-name-1 area may be either smaller than the data-name-2 area or larger than the data-name-2 area. In this case, extra memory is reserved to provide sufficient memory for the number of character positions in the largest of the redefining or redefined items.*

OSVS VSC2
MF

Note: The REDEFINES clause specifies the redefinition of a storage area, not the redefinition of the data items occupying the area.

3. Multiple redefinitions of a character position are permitted. The entries giving the new descriptions of the character positions must follow the defining entries without intervening entries that define new character positions. Multiple redefinitions of the same character positions must all use the data-name of the entry that originally defined the area *or the data-name of another preceding redefinition of that entry.*
4. The entries giving the new description of the character positions must not contain any VALUE clauses except in condition-name entries.
5. Multiple level 01 entries subordinate to any given level indicator (FD, CD, or SD) represent implicit redefinitions of the same area.

OSVS

Example

The following is an example of the REDEFINES clause:

```
05    A PICTURE X(6).
05    B REDEFINES A.
      10 B-1    PICTURE X(2).
      10 B-2    PICTURE 9(4).
05    C PICTURE 99V99.
```

In the foregoing 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.

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

Data items within an area can be redefined without their lengths being changed.

```

05  NAME-2.
    10  SALARY PICTURE XXX.
    10  SO-SEC-NO PICTURE X(9).
    10  MONTH PICTURE XX.
05  NAME-1 REDEFINES NAME-2.
    10  EMP-NO PICTURE X(6).
    10  WAGE PICTURE 999V999.
    10  YEAR PICTURE XX.

```

Data items can also be rearranged within an area.

```

05  B          PICTURE 99 USAGE DISPLAY VALUE 8.
05  C REDEFINES B PICTURE S99 USAGE COMPUTATIONAL.
05  A          PICTURE S99 USAGE COMPUTATIONAL.

```

Use of a redefining data item need not be the same as that of a redefined item. However, this does not change existing data.

```

05  REGULAR-EMPLOYEE.
    10  LOCATION PICTURE A(8).
    10  GRADE PICTURE X(4).
    10  SEMI-MONTHLY-PAY PICTURE 9999V99.
    10  WEEKLY-PAY REDEFINES SEMI-MONTHLY-PAY
        PICTURE 999V999.
05  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
    10  LOCATION PICTURE A(8).
    10  FILLER PICTURE X(6).
    10  HOURLY-PAY PICTURE 99V99.

```

The REDEFINES clause may be specified for an item within the scope of an area being redefined (that is, an item subordinate to a redefined item).

```

05  REGULAR-EMPLOYEE.
    10  LOCATION PICTURE A(8).
    10  GRADE PICTURE X(4).
    10  SEMI-MONTHLY-PAY PICTURE 999V999.
05  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
    10  LOCATION PICTURE A(8).
    10  FILLER PICTURE X(6).
    10  HOURLY-PAY PICTURE 99V99.
    10  CODE-H REDEFINES HOURLY-PAY PICTURE 9999.

```

The REDEFINES clause may also be specified for an item subordinate to a redefining item.

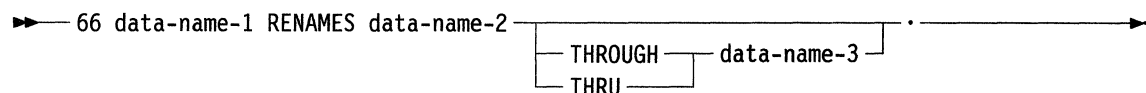
RENAMES Clause

Function

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary items.

General Format

The following figure shows the general format for the RENAMES clause:



Note: level-number 66 and data-name-1 are shown in the above format to improve clarity. level-number and data-name-1 are not part of the RENAMES clause.

Syntax Rules

The following syntax rules apply to the RENAMES clause:

1. Any number of RENAMES entries may be written for a logical record.
2. All RENAMES entries referring to data items in a logical record must immediately follow the last data description entry of the associated record description entry.
3. data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the same logical record. They cannot be the same data-name. A 66 level entry cannot rename another 66 level entry nor can it rename a 77, 88, or 01 entry.
4. data-name-1 cannot be used as a qualifier and can only be qualified by the names of the associated level 01, FD, CD, or SD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry or be subordinate to an item that has an OCCURS clause in its data description entry. Refer to “OCCURS Clause” on page 12-5.
5. The beginning of the area described by data-name-3 must not be to the left of the beginning of the area described by data-name-2. The end of the area described by data-name-3 must be to the right of the end of the area described by data-name-2. data-name-3, therefore, cannot be subordinate to data-name-2.
6. data-name-2 and data-name-3 may be qualified.
7. The words THRU and THROUGH are equivalent.
8. None of the items within the range, including data-name-2 and data-name-3, if specified, can be an item whose size is variable as defined in “OCCURS Clause” on page 12-5.

DBCS Support

9. data-name-1 may specify a USAGE DISPLAY-1 (DBCS) item if data-name-2 specifies a DBCS item and the THROUGH phrase is not specified.

End of DBCS Support

General Rules

The following general rules apply to the RENAMEs clause:

1. When data-name-3 is specified, data-name-1 is a group item that includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item). It concludes with data-name-3 (if data-name-3 is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).
2. When data-name-3 is not specified, all attributes of data-name-2 become the data attributes for data-name-1.

Example

The following is an example of valid and invalid RENAMEs specification:

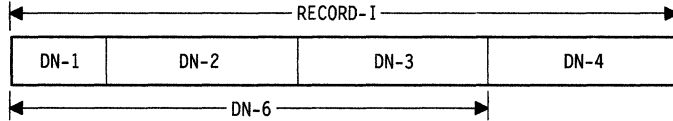
COBOL Specifications

Storage Layouts

Example 1 (Valid)

```

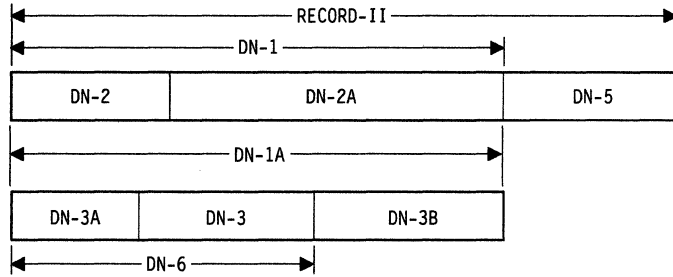
01 RECORD-I.
05 DN-1... .
05 DN-2... .
05 DN-3... .
05 DN-4... .
66 DN-6 RENAMES DN-1 THROUGH DN-3.
  
```



Example 2 (Valid)

```

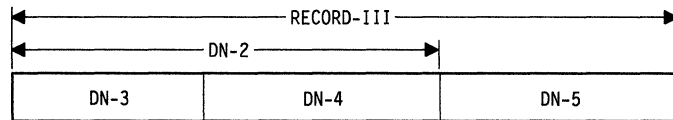
01 RECORD-II.
05 DN-1.
10 DN-2... .
10 DN-2A... .
05 DN-1A REDEFINES DN-1.
10 DN-3A... .
10 DN-3... .
10 DN-3B... .
05 DN-5... .
66 DN-6 RENAMES DN-2 THROUGH DN-3.
  
```



Example 3 (Invalid)

```

01 RECORD-III.
05 DN-2.
10 DN-3... .
10 DN-4... .
05 DN-5... .
66 DN-6 RENAMES DN-2 THROUGH DN-3.
  
```

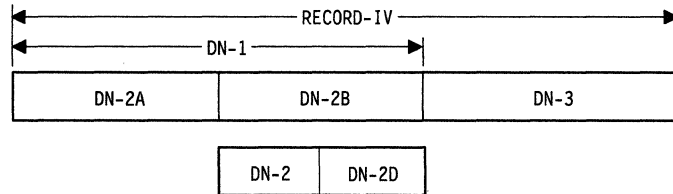


DN-6 is indeterminate

Example 4 (Invalid)

```

01 RECORD-IV.
05 DN-1.
10 DN-2A... .
10 DN-2B... .
10 DN-2C REDEFINES DN-2B.
15 DN-2... .
15 DN-2D... .
05 DN-3... .
66 DN-4 RENAMES DN-1 THROUGH DN-2.
  
```



DN-4 is indeterminate

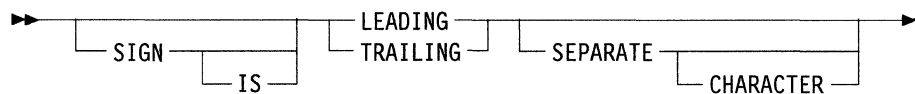
SIGN Clause

Function

The SIGN clause specifies the position and mode of representation of the operational sign when it is necessary to describe them explicitly.

General Format

The following figure shows the general format for the SIGN clause:



Syntax Rules

The following syntax rules apply to the SIGN clause:

1. The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character S or a group item containing at least one such numeric data description entry.
2. The numeric data description entries to which the SIGN clause applies must be described as USAGE IS DISPLAY.
3. If the CODE-SET clause is specified, signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause. *This restriction does not apply.* MF
Refer to "CODE-SET Clause" on page 8-40.

General Rules

The following general rules apply to the SIGN clause:

1. The optional SIGN clause specifies the position and the mode of representation of the operational sign for either the numeric data description entry to which it applies or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character S; the S indicates the presence of, but neither the representation nor, necessarily, the position of the operational sign.

-
2. A numeric data description entry whose PICTURE contains the character S, but to which no optional SIGN clause applies, has an operational sign. Neither the representation nor, necessarily, the position of the operational sign is specified by the character S. In the default case, general rules 3 through 5 do not apply to such signed numeric data items. The representation of the default operational sign is defined in "Selection of Character Representation and Radix" on page 2-20.
 3. If the optional SEPARATE CHARACTER phrase is not present, then:
 - a. The operational sign will be presumed to be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item in a manner defined in "Selection of Character Representation and Radix" on page 2-20.
 - b. The letter S in a PICTURE character string is not counted in determining the size of the item (in terms of standard data format characters).
 4. If the optional SEPARATE CHARACTER phrase is present, then:
 - a. The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.
 - b. The letter S in a PICTURE character string is counted in determining the size of the item (in terms of standard data format characters).
 - c. The operational signs for positive and negative are the standard data format characters + and -, respectively.
 5. Every numeric data description entry whose PICTURE contains the character S is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.
 6. If a SIGN clause is specified for an item (either an elementary numeric data description entry or a group item) that is subordinate to a group item for which a SIGN clause is also specified, then the SIGN clause specified in the subordinate item takes precedence.

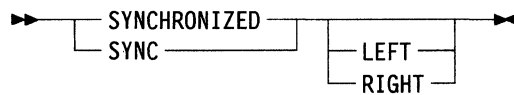
SYNCHRONIZED Clause

Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundaries of the computer memory.

General Format

The following figure shows the general format of the SYNCHRONIZED clause:



Syntax Rules

The following syntax rules apply to the SYNCHRONIZED clause:

1. This clause should only appear with an elementary item.
2. SYNC is an abbreviation for SYNCHRONIZED.
3. *The SYNCHRONIZED clause may appear with a nonelementary item.* OSVS VSC2

DBCS Support

4. The synchronized clause is ignored for a DBCS item.

End of DBCS Support

General Rules

The following general rules apply to the SYNCHRONIZED clause:

-
1. The effect of the SYNCHRONIZED clause is, by definition, implementation dependent.
 2. This clause specifies that the subject data item is to be aligned in the computer such that no other data item occupies any of the character positions between the leftmost and rightmost natural boundaries of the computer memory which delimits this data item. If the number of character positions required to store this data item is less than the number of character positions between those natural boundaries, the unused character positions (or portions thereof) must not be used for any other data item. However, such unused character positions are included in:
 - a. The size of any group item(s) to which the elementary item belongs
 - b. The character positions redefined when this data item is the object of a REDEFINES clause.

Thus, the size of an elementary item is unchanged by the SYNCHRONIZED clause, but extra character positions are assigned by the use of the clause.

3. SYNCHRONIZED not followed by either RIGHT or LEFT specifies that the elementary item is to be positioned between natural boundaries for efficient utilization of the elementary data item.
4. SYNCHRONIZED RIGHT specifies that the elementary item be positioned to terminate on the right character position of the natural boundary in which the elementary item is placed. *It only takes effect if the IBMCOMP system directive is set.* MF
5. Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause, is used in determining an action that depends on size, such as justification, truncation, or overflow.
6. If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.
7. When the SYNCHRONIZED clause is specified in a data description entry of a data item that also contains an OCCURS clause, or in a data description entry of a data item subordinate to a data description entry that contains an OCCURS clause, then:
 - a. Each occurrence of the data item is SYNCHRONIZED
 - b. An implicit FILLER generated for other data items within that same table is generated for each occurrence of those data items.
8. *If the SYNCHRONIZED clause is specified with a nonelementary item, the clause applies to all items subordinate to that nonelementary item.* OSVS VSC2
9. Specification of the LEFT phrase in the SYNCHRONIZED clause will have no effect. *Specification of the RIGHT phrase in the SYNCHRONIZED clause has an effect only if the IBMCOMP system directive is set.* MF

The effect of the use of the SYNCHRONIZED clause is discussed in "Selection of Character Representation and Radix" on page 2-20.

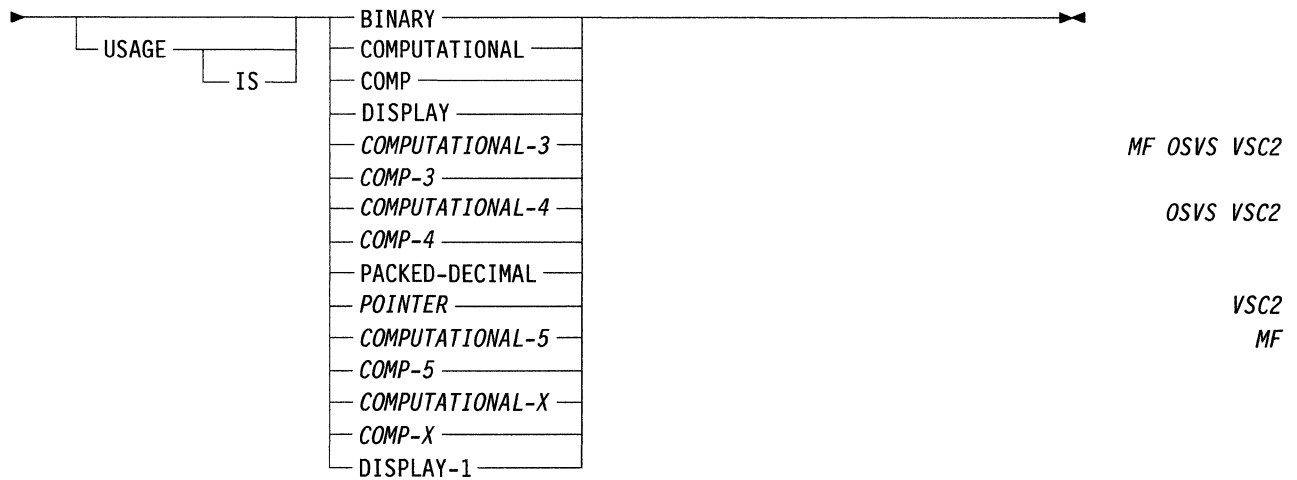
USAGE Clause

Function

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

General Format

The following figure shows the general format of the USAGE clause:



Syntax Rules

The following syntax rules apply to the USAGE clause:

1. The PICTURE character string of a COMPUTATIONAL, COMPUTATIONAL-3, or COMPUTATIONAL-5 item can contain only 9s, the operational sign character S, the implied decimal point character V, or one or more Ps. Refer to "PICTURE Clause" on page 6-18. VSC2 OSVS
MF
2. COMP is an abbreviation for COMPUTATIONAL.
3. BINARY is synonymous with COMPUTATIONAL.
4. COMP-4 is an abbreviation for COMPUTATIONAL-4 and is synonymous with COMPUTATIONAL. VSC2 OSVS
5. COMP-3 is an abbreviation for COMPUTATIONAL-3. MF
OSVS VSC2
6. PACKED-DECIMAL is synonymous with COMPUTATIONAL-3.
7. COMP-5 is an abbreviation for COMPUTATIONAL-5. MF
8. The USAGE clause cannot be specified for level 66 or 88 data description entries.

-
9. *The PICTURE character string of a COMPUTATIONAL-X item must consist either of all 9s or of all Xs. Either gives an item whose category is numeric.* MF
10. *The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE, SIGN, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS POINTER clause.* VSC2

DBCS Support

11. The DISPLAY-1 phrase defines an item as DBCS.

End of DBCS Support

General Rules

The following general rules apply to the USAGE clause:

1. The USAGE clause can be written at any level. If the USAGE clause is written at group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.
2. This clause specifies the manner in which a data item is represented in the memory of a computer. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.
3. A COMPUTATIONAL, *COMPUTATIONAL-3*, *COMPUTATIONAL-5*, or *COMPUTATIONAL-X* item is capable of representing a value to be used in computations and must be numeric. If a group item is described with one of these clauses, the clause applies to the elementary items in the group, not to the group item itself. The group item cannot be used in computations. The effect of these clauses is discussed in "Selection of Character Representation and Radix" on page 2-20. OSVS VSC2 MF
4. The USAGE IS DISPLAY clause indicates that the format of the data is a standard data format.
5. If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is implicitly DISPLAY.
6. Space requirements and storage definitions for the various USAGE storage options are given under "Selection of Character Representation and Radix" on page 2-20.
7. *The USAGE IS POINTER clause identifies a data item in which you can store the address of a data item (refer to "SET Statement" on page 7-84). USAGE IS POINTER can be specified for group items as well as for elementary items. However, only the elementary items within the group are regarded as pointer items; the group name cannot be used where a pointer name is expected. All of the elementary items in a group item with USAGE IS POINTER are treated as if they had the USAGE IS POINTER clause.* VSC2

VALUE Clause

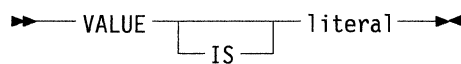
Function

The VALUE clause defines the value of constants, the initial value of working storage items, and the values associated with a condition name.

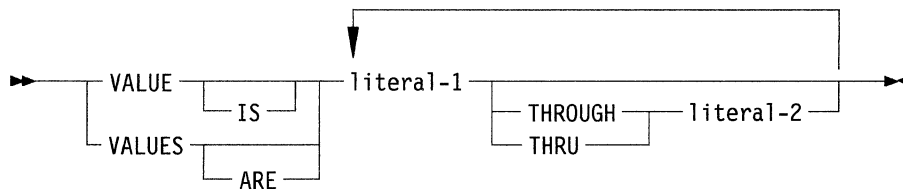
General Format

The following figures show the general format of the VALUE clause:

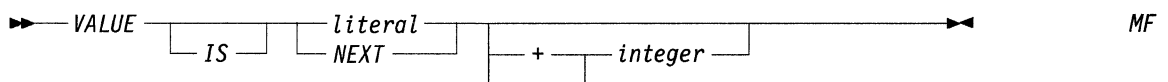
Format 1



Format 2



Format 3



Syntax Rules

The following syntax rules apply to the VALUE clause:

1. A signed numeric literal must have associated with it a signed numeric PICTURE character string.
2. All numeric literals in a VALUE clause of an item must have values which are within the range of values indicated by the PICTURE clause, and must not have a value that would require truncation of nonzero digits. Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.
3. The words THRU and THROUGH are equivalent.

DBCS Support

4. A VALUE clause associated with a DBCS item must contain a DBCS literal.

End of DBCS Support

General Rules

The following general rules apply to the VALUE clause:

1. The VALUE clause must not conflict with other clauses in the data description of the item or the data description within the hierarchy of the item. The following rules apply:
 - a. If the category of the item is a numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a working storage item, the literal is aligned in the data item according to the standard alignment rules. Refer to “Standard Alignment Rules” on page 2-19.
 - b. If the category of the item is alphabetic, alphanumeric, or alphanumeric-edited, all literals in the VALUE clause must be nonnumeric. The literal is aligned in the data item as if the data item had been described as alphanumeric. Refer to “Standard Alignment Rules” on page 2-19. Editing characters in the PICTURE clause are included in determining the size of the data item (refer to “PICTURE Clause” on page 6-18) but have no effect on initialization of the data item. Therefore, the VALUE for an edited item is presented in an edited form.
 - c. *If the item is numeric-edited, the value can be a numeric literal or a nonnumeric literal. If the value is a numeric literal, the value contained in the item will be the same as if the numeric literal were moved to the numeric-edited item.* MF
 - d. Initialization takes place independent of any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.
2. A figurative constant may be substituted in both Format 1 and Format 2 wherever a literal is specified.

DBCS Support

3. 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 may be used as DBCS literals.

The range of DBCS literals specified for the THROUGH phrase is based on the hexadecimal collating sequence.

End of DBCS Support

Condition-Name Rules

The following condition-name rules apply to the VALUE clause:

1. In a condition-name entry, the VALUE clause is required. The VALUE clause and the condition name itself are the only two clauses permitted in the entry. The characteristics of a condition name are implicitly those of its conditional variable.
2. Format 2 can be used only in connection with condition names. Wherever the THRU phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, and so on.

DBCS Support

3. Relation tests for DBCS items are performed according to the rules for comparison of DBCS items. These rules can be found in "Comparison of DBCS Operands" on page 7-11.

End of DBCS Support

Constant-Name Rules

The following is a list of constant-name rules:

MF

1. *The VALUE clause is required in a constant-name entry. The VALUE clause and the constant name itself are the only two clauses permitted in the entry. When the VALUE clause contains a literal, the characteristics of the constant name are implicitly those of the literal. When the VALUE clause contains NEXT, the constant name implicitly has the characteristics of an integer.*
2. *A constant name can be used in place of the literal or integer in a constant-name entry. Literal can be any kind of literal.*
3. *NEXT is a special register used by AIX VS COBOL system containing the address, relative to the start of the program, at which the next data item will start. NEXT can only be used in constant-name entries.*
4. *When used immediately before a data description entry that is preceded by slack bytes, or by bytes unused when a data area is redefined by a shorter area, NEXT contains the address where those bytes start. When used immediately after a description of a group item, NEXT contains the address where that item starts.*
5. *In the SCREEN SECTION, NEXT reflects implicit allocation of memory by the AIX VS COBOL system to handle the screen items described. The amount of memory allocated is the amount that would be allocated for data items with the same PICTURE string and USAGE DISPLAY. Because data descriptions in the LINKAGE SECTION do not cause memory to be allocated, NEXT is meaningless in the LINKAGE SECTION.*

Data Description Entries Other Than CONDITION-NAMES and CONSTANT-NAMES

The following data description entries apply to the VALUE clause:

1. Rules governing the use of the VALUE clause differ with the respective sections of the Data Division:
 - a. In the FILE SECTION, the VALUE clause must be used only in condition-name entries.
 - b. In the WORKING-STORAGE SECTION and the COMMUNICATION SECTION, the VALUE clause must be used in condition-name entries. The VALUE clause may also be used to specify the initial value of a data item. If it is, the clause causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in an item's description, the initial value of that data item is undefined.
 - c. In the LINKAGE SECTION, the value clause must be used in condition-name entries only.
2. *In the FILE SECTION and the LINKAGE SECTION, the VALUE clause may be used in data items entries, but is documentary only.* VSC2
3. The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause or that is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.
4. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause cannot be stated at the subordinate levels within this group.
5. The VALUE clause must not be written for a group containing items with descriptions, including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).
6. *The figurative constant NULL may be specified in the VALUE clause only if the data item is defined with USAGE POINTER. NULL is the only value you can specify in the VALUE clause for such an item. The effect is to set the pointer in such a way that it is guaranteed not to point to any data item.* VSC2
7. A Format 1 VALUE clause specified in a data description entry that contains an OCCURS clause, or in an entry that is subordinate to an OCCURS clause, causes every occurrence of the associated data item to be assigned the specified value.

Chapter 7. Procedure Division in the Nucleus

Contents

About This Chapter	7-5
Procedure Division in the Nucleus	7-6
General Format	7-6
Arithmetic Expressions	7-7
Definition of an Arithmetic Expression	7-7
Arithmetic Operators	7-8
Formation and Evaluation Rules	7-8
Conditional Expressions	7-9
Simple Conditions	7-9
Complex Conditions	7-15
Negated Simple Condition	7-16
Combined and Negated Combined Condition	7-16
Abbreviated Combined Relation Conditions	7-17
Condition Evaluation Rules	7-18
Common Phrases and General Rules for Statement Formats	7-19
ROUNDED Phrase	7-19
ON SIZE ERROR Phrase and NOT ON SIZE ERROR Phrase	7-19
CORRESPONDING Phrase	7-20
Arithmetic Statement Rules	7-20
Overlapping Operand Rules	7-21
Multiple Results in Arithmetic Statement Rules	7-21
Incompatible Data Rule	7-21
Signed Receiving Item Rule	7-21
ACCEPT Statement	7-22
Function	7-22
General Format	7-22
Syntax Rules	7-22
General Rules	7-23
Example	7-23
ADD Statement	7-24
Function	7-24
General Format	7-24
Syntax Rules	7-25
General Rules	7-25
Example	7-26
ALTER Statement	7-27
Function	7-27
General Format	7-27
Syntax Rules	7-27
General Rules	7-27
Example	7-28
COMPUTE Statement	7-29
Function	7-29
General Format	7-29
Syntax Rule	7-29
General Rules	7-29
Example	7-30
CONTINUE Statement	7-31
Function	7-31
General Format	7-31
Syntax Rule	7-31
General Rule	7-31
Example	7-31
DISPLAY Statement	7-32
Function	7-32
General Format	7-32
Syntax Rules	7-32
General Rules	7-33
Example	7-33

DIVIDE Statement	7-34
Function	7-34
General Format	7-34
Syntax Rules	7-36
General Rules	7-36
Example	7-37
ENTER Statement	7-38
Function	7-38
General Format	7-38
Syntax Rule	7-38
General Rule	7-38
EVALUATE Statement	7-39
Function	7-39
General Format	7-39
Syntax Rules	7-40
General Rules	7-40
Example	7-42
EXAMINE Statement	7-43
Function	7-43
General Format	7-43
Syntax Rules	7-43
General Rules	7-44
Example	7-44
EXEC(UTE) Statement	7-45
Function	7-45
General Format	7-45
Syntax Rule	7-45
General Rule	7-45
EXHIBIT Statement	7-46
Function	7-46
General Format	7-46
Syntax Rules	7-46
General Rules	7-47
Example	7-47
EXIT Statement	7-48
Function	7-48
General Format	7-48
Syntax Rules	7-48
General Rule	7-48
Example	7-49
GO TO Statement	7-50
Function	7-50
General Format	7-50
Syntax Rules	7-50
General Rules	7-51
IF Statement	7-52
Function	7-52
General Format	7-52
Syntax Rules	7-52
General Rules	7-52
INITIALIZE Statement	7-54
Function	7-54
General Format	7-54
Syntax Rules	7-54
General Rules	7-55
Example	7-56
INSPECT Statement	7-57
Function	7-57
General Format	7-57
Syntax Rules	7-59
General Rules	7-60
Examples	7-64

MOVE Statement	7-65
Function	7-65
General Format	7-65
Syntax Rules	7-65
General Rules	7-66
Example	7-68
MULTIPLY Statement	7-69
Function	7-69
General Format	7-69
Syntax Rules	7-70
General Rules	7-70
ON Statement	7-71
Function	7-71
General Format	7-71
Syntax Rules	7-71
General Rules	7-71
Example	7-72
PERFORM Statement	7-73
Function	7-73
General Format	7-73
Syntax Rules	7-74
General Rules	7-75
SET Statement	7-84
Function	7-84
General Format	7-84
Syntax Rules	7-84
General Rules	7-85
STOP Statement	7-86
Function	7-86
General Format	7-86
Syntax Rules	7-86
General Rules	7-86
STRING Statement	7-87
Function	7-87
General Format	7-87
Syntax Rules	7-87
General Rules	7-88
Example	7-90
SUBTRACT Statement	7-91
Function	7-91
General Format	7-91
Syntax Rules	7-92
General Rules	7-92
TRANSFORM Statement	7-94
Function	7-94
General Format	7-94
Syntax Rules	7-94
General Rules	7-95
Example	7-95
UNSTRING Statement	7-96
Function	7-96
General Format	7-96
Syntax Rules	7-96
General Rules	7-97
Example	7-101

About This Chapter

This chapter describes the Procedure Division in the nucleus. It explains the format, syntax, and general rules of Procedure Division statements and phrases. Arithmetic expressions, conditional expressions, and statement format rules also are covered.

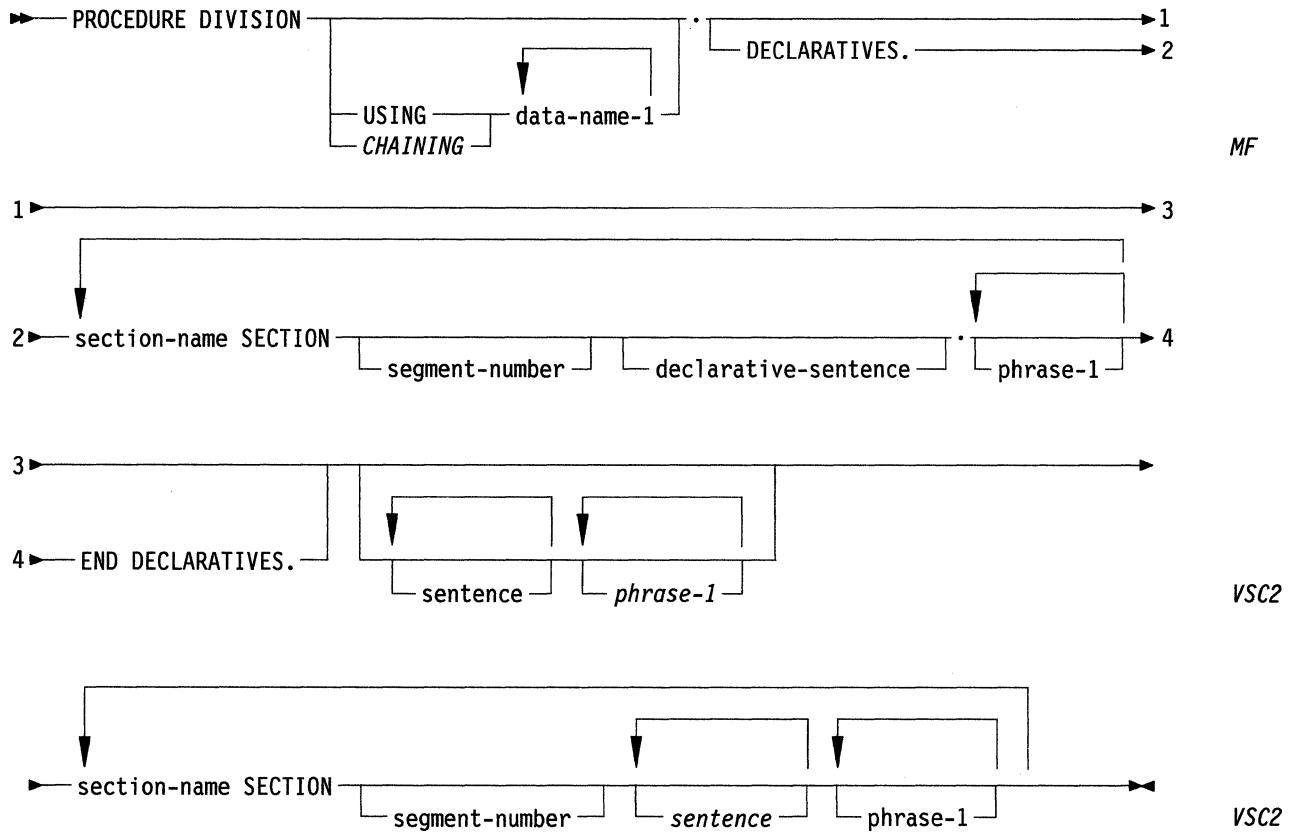
Procedure Division in the Nucleus

This section describes the Procedure Division in the nucleus.

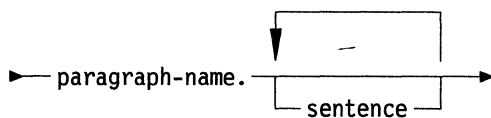
General Format

The following figures show examples of the two general formats of the Procedure Division:

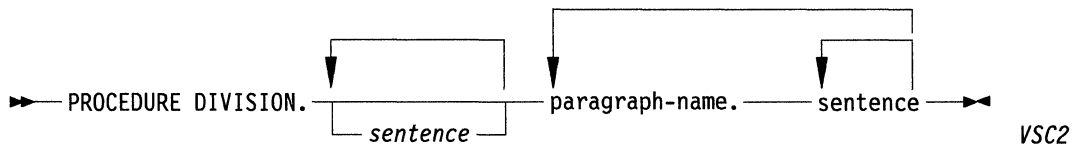
Format 1



where phrase-1 is:



Format 2



The USING phrase of the Procedure Division header is specified only in programs that are to be CALLED or CHAINED.

MF

Refer to Chapter 11, "Interprogram Communication." The declarative sentence shown in "General Format" on page 7-6 is a USE statement (refer to Chapter 5, "Environment Division in the Nucleus," Chapter 9, "COBOL Source Library," and Chapter 14, "Report Writer"). It specifies when the section is to be executed. *However, a section in the declarative section need not have a declarative sentence: it can be invoked by a PERFORM statement (refer to "PERFORM Statement" on page 7-73, and "PERFORM Statement" on page A-9, for details of the PERFORM statement).*

MF

Arithmetic Expressions

This section describes the arithmetic expressions associated with the Procedure Division.

Definition of an Arithmetic Expression

An arithmetic expression can be:

- an identifier of a numeric elementary item
- a numeric literal
- identifiers and literals separated by arithmetic operators
- two arithmetic expressions separated by an arithmetic operator, or
- an arithmetic expression enclosed in parentheses.

Any arithmetic expression may be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operator, and parentheses are given in Table 7-1 on page 7-8.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

FIRST SYMBOL	SECOND SYMBOL			
	Variable	* / ** + -	Unary + , -	()
Variable	-	P	-	P
* / ** + -	P	-	P	-
Unary + , -	P	-	-	-
(P	-	P	-
)	-	P	-	P

Note:

P indicates a permissible pair of symbols.
- indicates an invalid pair.
Variable indicates an identifier or literal.

Arithmetic Operators

There are five binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that may be preceded by a space and followed by a space. The arithmetic operators are defined as follows:

Binary Arithmetic Operators

	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Arithmetic Operators

	Meaning
+	The effect of multiplication by numeric literal +1
-	The effect of multiplication by numeric literal -1

Formation and Evaluation Rules

The following formation and evaluation rules apply to arithmetic expressions:

1. Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first. Within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

- 1st Unary plus and minus
- 2nd Exponentiation
- 3rd Multiplication and division
- 4th Addition and subtraction

-
2. Parentheses are used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.
 3. An arithmetic expression may only begin with the symbol left parentheses, plus, or minus, or a variable. It may only end with a right parentheses or a variable. There must be a one-to-one correspondence between left and right parenthesis of an arithmetic expression so that each left parenthesis is to the left of its corresponding right parenthesis.
 4. Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite operands or receiving data items. Refer to syntax rule 3 of "ADD Statement" on page 7-24.

Conditional Expressions

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. Conditional expressions are specified in the EVALUATE, IF, PERFORM, and SEARCH statements. There are two categories of conditions associated with conditional expressions:

- simple conditions
- complex conditions.

Each may be enclosed within any number of paired parentheses. If it is, its category is not changed.

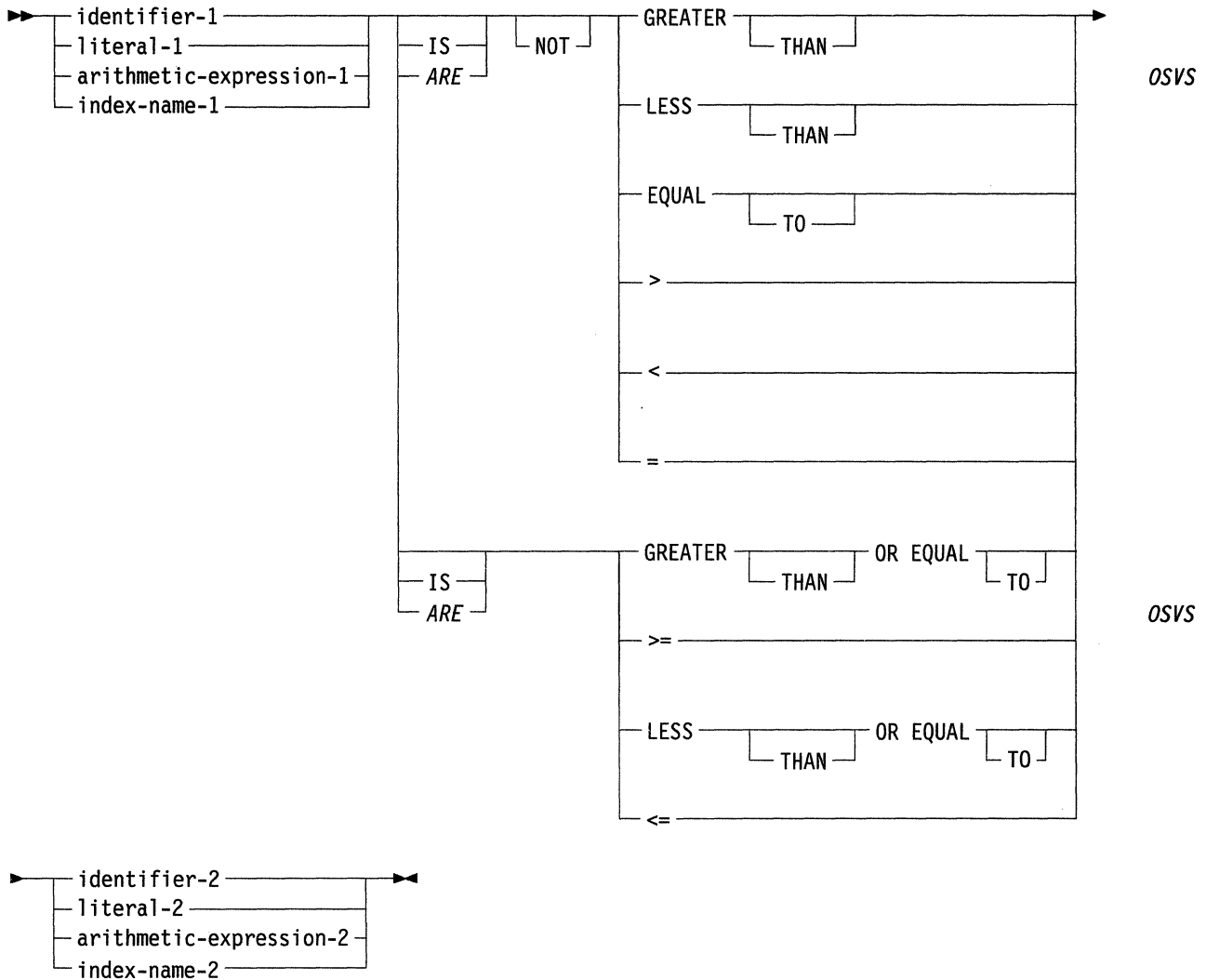
Simple Conditions

The simple conditions are the relation, class, condition-name, switch-status, and sign conditions. A simple condition has a truth value of true or false. The inclusion in parentheses of simple conditions does not change the simple truth value.

Relation Condition

A relation condition causes a comparison of two operands, each of which may be the data item referred to by an identifier, a literal, or the value resulting from an arithmetic expression. A relation condition has a truth value of true if the relation exists between the operands. Comparison of two numeric operands is permitted regardless of the formats specified in their respective USAGE clauses. However, for all other comparisons the operands must have the same usage. If either of the operands is a group item, the nonnumeric comparison rules apply.

The following figure shows the general format of a relation condition.



The first operand (`identifier-1`, `literal-1`, `index-name-1`, or `arithmetic-expression-1`) is called the subject of the condition; the second operand (`identifier-2`, `literal-2`, `index-name-2`, or `arithmetic-expression-2`) is called the object of the condition. The relation condition must contain at least one reference to a variable.

The relational operator specifies the type of comparison to be made in a relation condition. A space must precede and follow each reserved word comprising the relational operator. When used, `NOT` and the next key word or relation character are one relational operator that defines the comparison to be executed for truth value. For example:

- `NOT EQUAL` is a truth test for an unequal comparison
- `NOT GREATER` is a truth test for an equal or less comparison.

The meaning of the relational operators is as shown in Figure 7-1.

DBCS Support

Comparison of DBCS Operands

Double-Byte Character Set (DBCS) data items and literals can be used with all relational operators. Comparisons are based on the hexadecimal representations of the DBCS characters. The rules for comparing DBCS operands are the same as those for comparing nonnumeric operands. (The PROGRAM COLLATING SEQUENCE clause will not be applied in comparisons of DBCS data items and literals.) If the DBCS items are not the same length, the smaller item is padded on the right with DBCS spaces.

End of DBCS Support

Meaning	Relational Operator
Greater than or not greater than	
Less than or not less than	
Equal or not equal to	
Greater than or equal to	
Less than or equal to	

Figure 7-1. Relational Operators

Comparison of Numeric Operands

For operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands. The length of the literal or arithmetic expression operands, in terms of number of digits represented, is not significant. Zero is considered a unique value, regardless of the sign.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparison.

Comparison of Nonnumeric Operands

For nonnumeric operands or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters. Refer to "OBJECT-COMPUTER Paragraph" on page 5-6. If one of the operands is specified as numeric, it must be an integer data item or an integer literal and:

1. If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric-data item (in terms of standard data format characters). The contents of this alphanumeric data item were then compared to the nonnumeric operand. Refer to "MOVE Statement" on page 7-65 and "Symbols Used" on page 6-21.
2. If the nonnumeric operand is a group item, the numeric operand is treated as though it was moved to a group item of the same size as the the numeric data item (in terms of standard data format characters). The contents of this group item were then compared to the nonnumeric operand. Refer to "MOVE Statement" on page 7-65 and "Symbols Used" on page 6-21.
3. A non-integer numeric operand cannot be compared to a nonnumeric operand.

The size of an operand is the total number of standard data format characters in the operand.

Numeric and nonnumeric operands may be compared when their usage is not the same. *MF*

- If the operands are of equal size, comparison effectively proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of characters compare equally through the last pair, when the low order end is reached.

The first encountered pair of unequal characters is compared to determine their relative position in the collating sequence. The operand that contains the character that is positioned higher in the collating sequence is considered to be the greater operand.

- If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right a sufficient number of spaces to make the operands of equal size.

Comparisons Involving Index-Names and/or Index Data Items

Relation tests may be made only between the following:

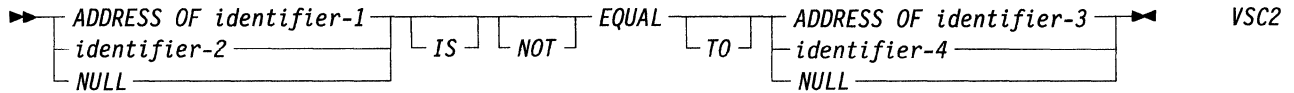
1. Two index-names. The result is the same as if the corresponding occurrence numbers were compared.
2. An index-name and a numeric data item (other than an index data item) or numeric literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

3. An index data item and an index-name or another index data item. The actual values are compared without conversion.

Comparisons Involving Data Items with USAGE POINTER

Two items whose USAGE is either implicitly or explicitly POINTER can be compared. Only the relational operators EQUAL and NOT EQUAL are permitted in pointer comparisons. VSC2

The format of a pointer comparison is as follows:



where:

identifier-1 and identifier-3 refer to 01 or 77 level items in the LINKAGE or WORKING-STORAGE SECTION. MF

identifier-2 and identifier-4 refer to items with USAGE IS POINTER.

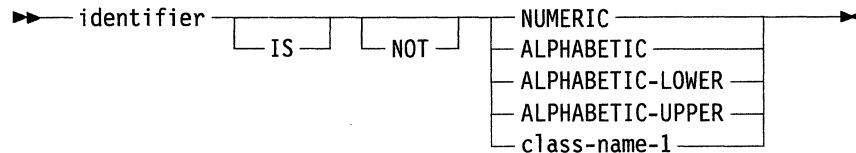
Note: Comparison is not allowed in which both operands are the figurative constant NULL.

The operands are equal if the two addresses are the same. Otherwise, they are unequal.

This type of relation condition is allowed in IF, PERFORM, EVALUATE, and SEARCH (Format 1) statements. It is not allowed in SEARCH (Format 2) statements (SEARCH ALL) because no meaningful ordering can be applied to pointer data items.

Class Condition

The class condition determines whether the operand is numeric (consisting entirely of the characters 0 through 9), with or without the operational sign, or alphabetic (consisting entirely of the characters A, B, C, ... Z, space (uppercase alphabetic letters only)). Following is the general format for the class condition:



The usage of the operand being tested must be described either as DISPLAY or, in the case of the NUMERIC test, as DISPLAY, COMPUTATIONAL, COMPUTATIONAL-3, COMPUTATIONAL-5, or COMPUTATIONAL-X. OSVS VSC2 MF

When used, NOT and the next key word specify one class condition that defines the class test to be executed for truth value; for example, NOT NUMERIC is a truth test for determining that an operand is nonnumeric.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operation sign(s). If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present. If the data description of an elementary item being tested does indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and a valid operational sign is present.

Valid operational signs for data items described with the SIGN IS SEPARATE clause are the standard data format characters + and -. Valid operational signs for data items not described with the SIGN IS SEPARATE clause are described in "Selection of Character Representation and Radix" on page 2-20.

The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the uppercase alphabetic characters A through Z and the space, or any combination of the lowercase alphabetic characters a through z and the space.

The ALPHABETIC-LOWER test cannot be used with an item whose data description describes the item as numeric. The result of the test is true if the content of the data item referenced by identifier-1 consists entirely of the lowercase alphabetic characters a through z and space.

The ALPHABETIC-UPPER test cannot be used with an item whose data description describes the item as numeric. The result of the test is true if the content of the data item referenced by identifier-1 consists entirely of the uppercase alphabetic characters A through Z and space.

The class-name-1 test must not be used with an item whose data description describes the item as numeric.

Condition-Name Condition (Conditional Variable)

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition-name. The general format for the condition-name condition is shown in the following example:

condition-name

If the condition-name is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

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.

_____ DBCS Support _____

Condition-names with DBCS values are allowed.

_____ End of DBCS Support _____

Switch-Status Condition

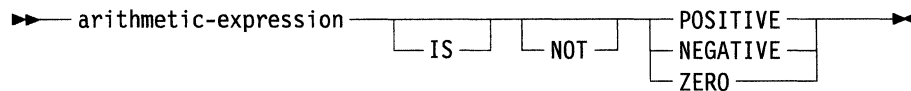
A switch-status condition determines the on or off status of one of the nine COBOL switches named, respectively, SWITCH 0 through SWITCH 8. The value of each of these switches (on or off) is determined by the operator at the beginning of execution of the COBOL object program. Refer to the *User's Guide*. The switch and the on or off value associated with the condition must be named in the SPECIAL-NAMES paragraph of the Environment Division. The general format for the switch-status condition is as follows:

condition-name

The result of the test is true if the switch is set to the specified position corresponding to the condition-name.

Sign Condition

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to zero. The general format for a sign condition is as follows:



When used, NOT and the next key word specify one sign condition that defines the algebraic test to be executed for truth value. For example, NOT ZERO is a truth test for a nonzero (positive or negative) value. An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero. The arithmetic expression must contain at least one reference to a variable.

Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions, and/or complex conditions with logical connectors (logical operators AND and OR), or negating these conditions with logical negation (the logical operator NOT). The truth value of a complex condition, whether parenthesized or not, is the result of the interaction of all the stated logical operators on the individual truth values of simple conditions, or the intermediate truth values of conditions logically connected or logically negated.

The following list shows the logical operators and their meanings:

Logical Operator	Meaning
AND	Logical conjunction; the truth value is true if both of the conjoined conditions are true; false if one or both of the conjoined conditions is false.
OR	Logical inclusive OR; the truth value is true if one or both of the included conditions is true; false if both included conditions are false.
NOT	Logical negation or reversal of truth value; the truth value is true if the condition is false; false if the condition is true.

The logical operators must be preceded by a space and followed by a space.

Negated Simple Condition

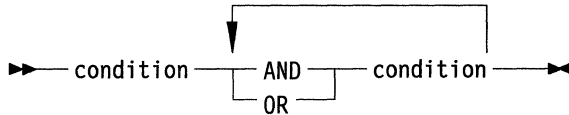
A simple condition is negated using the logical operator NOT. The negated simple condition effects the opposite truth value for a simple condition. Thus, the truth value of a negated simple condition is true if and only if the truth value of the simple condition is false; the truth value of a negated simple condition is false if and only if the truth value of the simple condition is true. The inclusion in parentheses of a negated simple condition does not change the truth value.

Following is the general format for a negated simple condition:

► NOT simple-condition ◄

Combined and Negated Combined Condition

A combined condition results from connecting conditions with one of the logical operators AND or OR. The following figure shows the general format for a combined condition.



where condition may be:

- A simple condition
- A negated simple condition
- A combined condition
- A negated combined condition (the NOT logical operator followed by a combined condition enclosed within parentheses)
- Combinations of the preceding conditions, specified according to the rules summarized in Table 7-2 on page 7-17.

Although parentheses need never be used when either AND or OR (but not both) is used exclusively in a combined condition, parentheses may be used to effect a final truth value when a mixture of AND, OR and NOT is used.

In the absence of the relevant parenthesis in a complex condition, the precedence (binding power) of the logical operators determines the conditions to which the specified logical operators apply and implies the equivalent parenthesis. The order of precedence is NOT, AND, OR. By specifying condition-1 OR NOT condition-2 AND condition-3 it implies and is equivalent to specifying condition-1 OR ((NOT condition-2) AND condition-3).

Where parentheses are used in a complex condition, precedence is used to determine the binding of conditions to logical operator. Therefore, parenthesis can be used to depart from the normal precedence of logical operators as specified above. Thus, the example complex condition above can be given a different meaning by specifying it as (condition-1 OR (NOT condition-2)) AND condition-3. Refer to "Condition Evaluation Rules" on page 7-18.

Table 7-2 on page 7-17 shows the ways in which conditions and logical operators may be combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses so that each left parenthesis is to the left of its corresponding right parenthesis.

Element	Permitted Location in Conditional Expression	Element Can Be Preceded by Only:	Element Can Be Followed by Only:
simple-condition	Any	OR, NOT, AND, (OR, AND,)
OR, or AND	Not first or last	simple-condition,)	simple-condition, NOT, (
NOT	Not last	OR, AND, (simple-condition, (
(Not last	OR, NOT, AND, (simple-condition, NOT, (
)	Not first	simple-condition,)	OR, AND,)

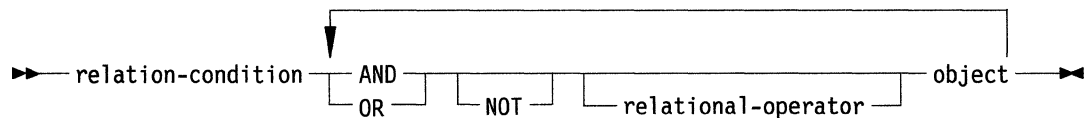
The element pair OR NOT is permissible while the pair NOT OR is not permissible. NOT (is permissible while NOT NOT is not permissible.

Abbreviated Combined Relation Conditions

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence so that a succeeding relation condition contains a subject or a subject and a relational operator that is common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first may be abbreviated by the following:

- The omission of the subject of the relation condition
- The omission of the subject and the relational operator of the relation condition.

The format for an abbreviated combined relation condition is:



Within a sequence of relation conditions both of the above forms of abbreviation may be used. The effect of using such abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The result of such implied insertion must comply with the rules of Table 7-2. This insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

The order of evaluation of the conditions can be prioritized by using parentheses (see example below). OSVS

The interpretation applied to the use of the word NOT in an abbreviated combined relation condition is as follows:

1. If the word immediately following NOT is GREATER, >, LESS, <, EQUAL, or =, then the NOT participates as part of the relational operator.
2. Otherwise, the NOT is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

Following are some examples of abbreviated-combined and negated-combined relation conditions and expanded equivalents:

Abbreviated Combined Relation Condition	Expanded Equivalent
$a > b \text{ AND NOT } < c \text{ OR } d$	$((a > b) \text{ AND } (a \text{ NOT } < c)) \text{ OR } (a \text{ NOT } < d)$
$a \text{ NOT EQUAL } b \text{ OR } c$	$(a \text{ NOT EQUAL } b) \text{ OR } (a \text{ NOT EQUAL } c)$
$\text{NOT } a = b \text{ OR } c$	$(\text{NOT}(a = b)) \text{ OR } (a = c)$
$\text{NOT}(a \text{ GREATER } b \text{ OR } < c)$	$\text{NOT}((a \text{ GREATER } b) \text{ OR } (a < c))$
$\text{NOT}(a \text{ NOT } > b \text{ AND } c \text{ AND NOT } d)$	$\text{NOT}(((a \text{ NOT } > b) \text{ AND } (a \text{ NOT } > c)) \text{ AND } (\text{NOT}(a \text{ NOT } > d))))$
$x > a \text{ OR } y \text{ AND } z$	$x > a \text{ OR } (x > y \text{ AND } x > z)$
$x > a \text{ OR } (y \text{ AND } z)$	$x > a \text{ OR } (x > y \text{ AND } x > z)$ OSVS
$x > (a \text{ OR } y) \text{ AND } z$	$(x > a \text{ OR } x > y) \text{ AND } x > z$
$x (= a \text{ OR } > b)$	$x = a \text{ OR } x > b$
$x = a \text{ AND } (> b \text{ OR } < z)$	$x = a \text{ AND } (x > b \text{ OR } x < z)$

Condition Evaluation Rules

Parentheses may be used to specify the order in which individual conditions of complex conditions are to be evaluated when it is necessary to depart from the implied evaluation precedence.

Conditions within parentheses are evaluated first. Within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition. When parentheses are not used, or parenthesized conditions are at the same level of inclusiveness, the following hierarchical order of logical evaluation is implied until the final truth value is determined:

1. Values are established for arithmetic expressions. Refer to "Arithmetic Expressions" on page 7-7.
2. Truth values for simple conditions are established in the following order:
 - relation (following the expansion of any abbreviated relation condition)
 - class
 - condition-name
 - switch-status
 - sign.
3. Truth values for negated conditions are established.
4. Truth values for combined conditions are established: AND logical operators, followed by OR logical operators.
5. Truth values for negated combined conditions are established.
6. When the sequence of evaluation is not completely specified by parentheses, the order of evaluation of consecutive operations of the same hierarchical level is from left to right.

Common Phrases and General Rules for Statement Formats

In the statement descriptions that follow, several phrases appear frequently:

- **ROUNDED** phrase
- **ON SIZE ERROR** phrase
- **NOT ON SIZE ERROR** phrase
- **CORRESPONDING** phrase.

These phrases are described in the following sections:

ROUNDED Phrase

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. A resultant-identifier is that identifier associated with the result of an arithmetic operation. When rounding is requested, the absolute value of the resultant-identifier is increased by one whenever the most significant digit of the excess is greater than or equal to five.

When the low-order integer positions in a resultant-identifier are represented by the character P in the PICTURE for the resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which memory is allocated.

ON SIZE ERROR Phrase and NOT ON SIZE ERROR Phrase

If, after decimal point alignment, the absolute value of a result of an arithmetic operation exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results, except in **MULTIPLY** and **DIVIDE** statements. In this case the size error condition applies to the intermediate results as well. If the **ROUNDED** phrase is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the **SIZE ERROR** phrase is specified.

ON SIZE ERROR Phrase Not Specified

When a size error condition occurs, the value of the resultant-identifier(s) affected is undefined. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

The **NOT ON SIZE ERROR** phrase, if specified, is ignored, and the imperative statement associated with it is not executed.

ON SIZE ERROR Phrase Specified

When a size error condition occurs, then the values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation. After completion of the execution of this operation, the imperative statement in the **SIZE ERROR** phrase is executed.

The **NOT ON SIZE ERROR** phrase, if specified, is ignored, and the imperative statement associated with it is not executed.

NOT ON SIZE ERROR Phrase

If the NOT ON SIZE ERROR phrase is specified for an arithmetic operation statement, and after execution of that statement a size error condition (as defined above) does not exist, then the NOT ON SIZE ERROR phrase, if specified, is executed. The ON SIZE ERROR phrase, if specified, is ignored, and the imperative statement associated with it is not executed.

For the ADD statement with the CORRESPONDING phrase and the SUBTRACT statement with the CORRESPONDING phrase, if any of the individual operations produces a size error condition, the imperative statement in the ON SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.

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, an implicit transfer or control is made after execution of the phrase to the end of the arithmetic statement.

CORRESPONDING Phrase

In the following list d1 and d2 must each be identifiers that refer to group items. A pair of data items, one from d1 and one from d2, correspond if the following conditions exist:

1. A data item in d1 and a data item in d2 are not designated by the key word FILLER and have the same data-name and the same qualifiers up to, but not including, d1 and d2.
2. At least one of the data items is an elementary data item in the case of a MOVE statement with the CORRESPONDING phrase; and both of the data items are elementary numeric data items in the case of the ADD statement with the CORRESPONDING phrase or the SUBTRACT statement with the CORRESPONDING phrase.
3. The description of d1 and d2 must not contain level-number 66, 77, 78, *MF* or 88 or the USAGE IS INDEX clause.
4. A data item that is subordinate to d1 or d2 and contains a REDEFINES, OCCURS, USAGE IS INDEX, or *USAGE IS POINTER* clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, OCCURS, USAGE IS INDEX or *USAGE IS POINTER* clause. However, d1 and d2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses. *VSC2*
VSC2

Arithmetic Statement Rules

ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT are arithmetic statements. Common features are as follows:

- The data descriptions of the operands need not be the same because any necessary conversions and decimal point alignments are supplied throughout the calculation.
- The maximum size of each operand is 18 decimal digits. The composite of operands, which is a hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points must not contain more than 18 decimal digits. Refer to "ADD Statement" on page 7-24, "DIVIDE Statement" on page 7-34, "MULTIPLY Statement" on page 7-69, and "SUBTRACT Statement" on page 7-91.

Overlapping Operand Rules

When a sending and a receiving item in an arithmetic statement or an INSPECT, MOVE, SET, STRING, or UNSTRING statement share a part of their memory areas, the result of the execution of the statement is undefined.

Multiple Results in Arithmetic Statement Rules

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple results. Such statements respond as though they had been written in the following way:

1. A statement that accesses all data items which are part of the initial evaluation of the statement performs all arithmetic necessary to arrive at the result to be stored in the receiving items and stores that result in a temporary memory location.
2. A sequence of statements that transfer or combine the value of a temporary memory location with each single resultant data item are considered to be written in the same left-to-right sequence in which the multiple results are listed.

The result of the statement:

```
ADD a, b, c TO c, d (c), e
```

is equivalent to:

```
ADD a, b, c GIVING temp
ADD temp TO c
ADD temp TO d (c)
ADD temp TO e
```

and the result of the statement:

```
MULTIPLY a(i) BY i,a(i)
```

is equivalent to:

```
MOVE a(i) to temp
MULTIPLY temp by i
MULTIPLY temp by a(i)
```

where temp is an intermediate result item provided by the AIX VS COBOL system.

Incompatible Data Rule

Except for the class condition (refer to "Class Condition" on page 7-13), when the contents of a data item are referred to in the Procedure Division and the contents of that data item are not compatible with the class specified for that data item by its PICTURE clause, the result of such a reference is undefined.

Signed Receiving Item Rule

When the receiving item in an arithmetic statement or a MOVE statement is a signed numeric or a signed numeric-edited item, the sign is moved into the receiving item independently of any truncation of the absolute numeric data. It is possible, therefore, for the numeric value to be zero but for the sign to be negative.

ACCEPT Statement

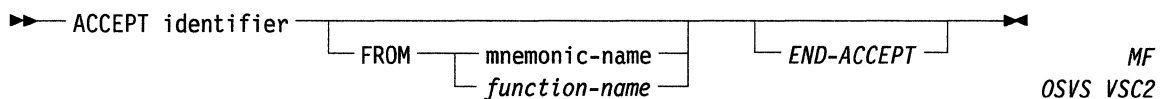
Function

The ACCEPT statement causes data keyed at the console or supplied by the operating system to be made available to the program in a specified data item. Refer to Chapter 18, "Screen-Handling" for additional formats of the ACCEPT statement.

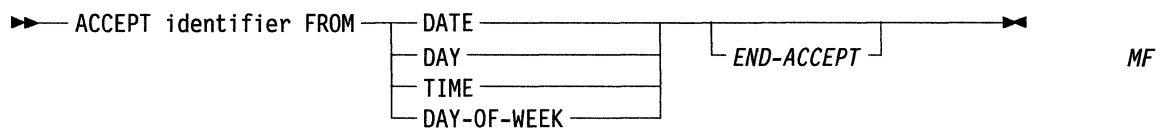
General Format

The following figure shows the general format of the ACCEPT statement:

Format 1



Format 2



Syntax Rules

The following syntax rules apply to the ACCEPT statement:

Format 1

1. The mnemonic-name in Format 1 must be associated with a function name in the SPECIAL NAMES paragraph in the Environment Division. Refer to rule 9 on page 5-13 under "SPECIAL-NAMES Paragraph" on page 5-8 for a list of valid function-names.
2. *Alternatively, function-name can itself be used instead of an associated mnemonic-name.* OSVS
3. Each literal may be any figurative constant, except ALL.

Format 2

4. The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier according to the rules of the MOVE statement. DATE, DAY, and TIME are conceptual data items and, therefore, are not described in the COBOL program.

General Rules

The following general rules apply to the ACCEPT statement:

Format 1

1. When the operand of an ACCEPT statement has USAGE other than DISPLAY or an individual sign, conversion to the correct format takes place after data has been transferred into a temporary area according to 2.
2. *If the function-name COMMAND-LINE or a mnemonic name associated with the function-name COMMAND-LINE is specified, data is overwritten by the contents of a system-dependent command-line buffer. All other permissible function-names are treated as equivalent to CONSOLE. Refer to "SPECIAL-NAMES Paragraph" on page 18-9.* *MF*

Format 2

3. The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier according to the rules of the MOVE statement. DATE, DAY, and TIME are conceptual data items and, therefore, are not described in the COBOL program.
4. DATE is composed of the data elements year of century, month of year, and day of month. The sequence of the data element codes shall be from high to low order (left to right), year of century, month of year, and day of month. Date, when accessed by a COBOL program, responds as if it had been described in the COBOL program as an unsigned elementary numeric integer data item six digits in length.
5. DAY is composed of the data elements year of century and day of year. The sequence of the data element codes shall be from high order to low order (left to right), year of century, day of year. Therefore, July 1, 1968, would be expressed as 68183. DAY, when accessed by a COBOL program, responds as if it had been described in a COBOL program as an unsigned elementary numeric integer data item five digits in length.
6. TIME is composed of the data elements hours, minutes, seconds, and hundredths of a second. TIME is based on elapsed time after midnight on a 24-hour clock basis (2:41 P.M. would be expressed as 14410000). TIME, when accessed by a COBOL program, responds as if it had been described in a COBOL program as an unsigned elementary numeric integer data item eight digits in length. The minimum value of TIME is 00000000; the maximum value of TIME is 23595999. If the hardware does not have the facility to provide fractional parts of TIME, the value is converted to the closest decimal approximation.
7. DAY-OF-WEEK is composed of a single data element whose content represents the day of the week. DAY-OF-WEEK behaves as an unsigned elementary numeric integer data item, 1 digit in length. In DAY-OF-WEEK, the value 1 represents Monday, 2 represents Tuesday, ... 7 represents Sunday.

Example

The following example shows the ACCEPT statement:

```
ACCEPT myval FROM COMMAND-LINE.  
ACCEPT CURRENT-DATE FROM DATE.
```

ADD Statement

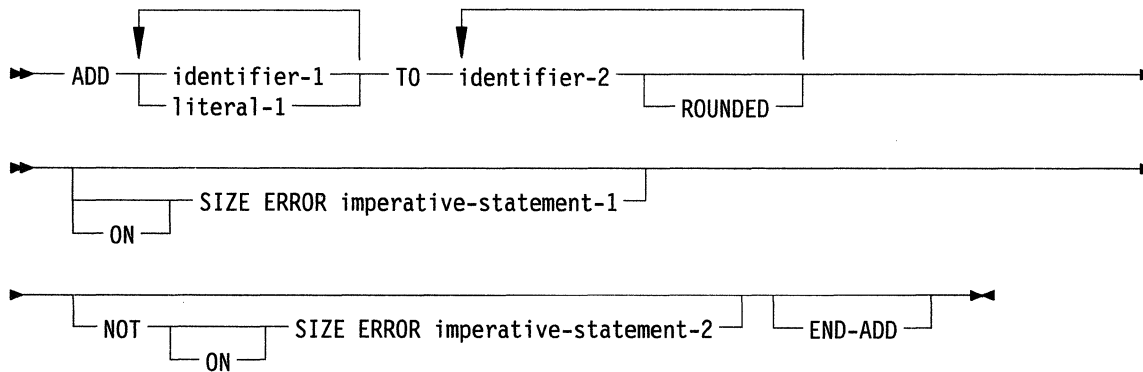
Function

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

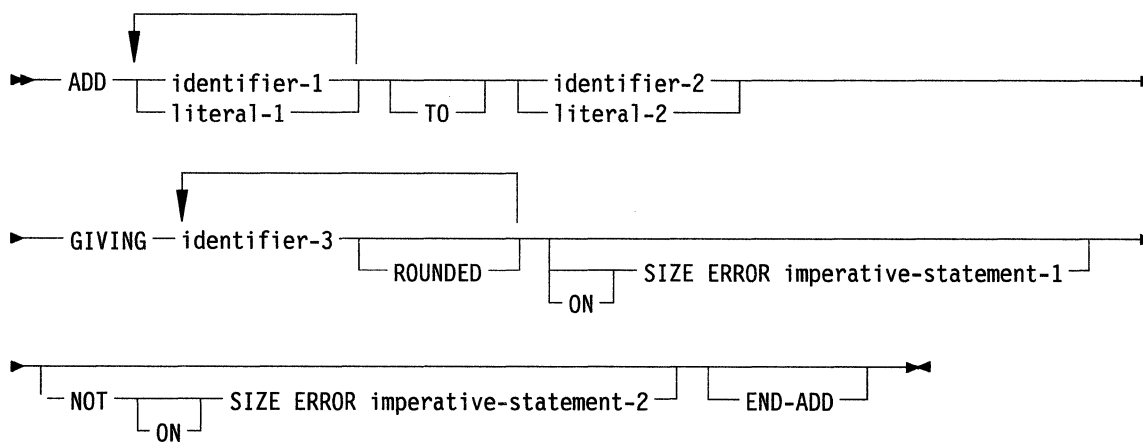
General Format

The following figures show the general format for the ADD statement:

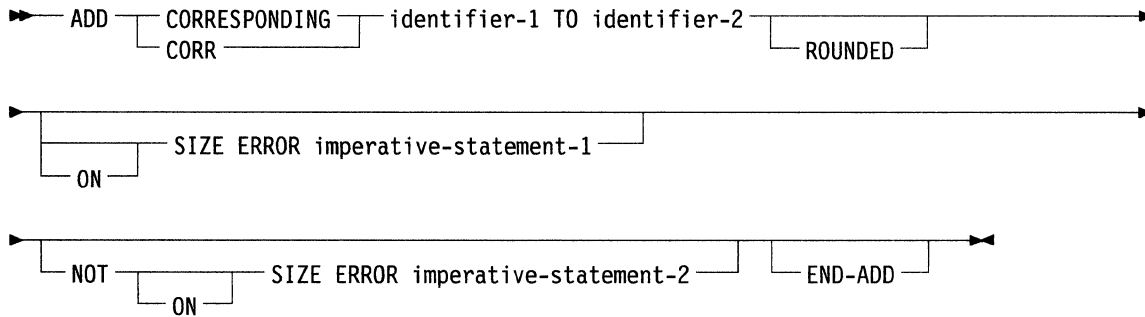
Format 1



Format 2



Format 3



Syntax Rules

The following syntax rules apply to the ADD statement:

1. In Formats 1 and 2, each identifier must refer to an elementary numeric item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric-edited item. In Format 3, each identifier must refer to a group item.
2. Each literal must be a numeric literal.
3. The composite of operands must not contain more than 18 digits. Refer to "Arithmetic Statement Rules" on page 7-20.
 - a. In Format 1 the composite of operands is determined by using all of the operands in a given statement.
 - b. 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.
 - c. In Format 3 the composite of operands is determined separately for each corresponding pair of data items.
4. CORR is an abbreviation for CORRESPONDING.

General Rules

The following general rules apply to the ADD statement:

1. Refer to "ROUNDED Phrase" on page 7-19, "ON SIZE ERROR Phrase and NOT ON SIZE ERROR Phrase" on page 7-19, "CORRESPONDING Phrase" on page 7-20, "Arithmetic Statement Rules" on page 7-20, "Overlapping Operand Rules" on page 7-21, "Multiple Results in Arithmetic Statement Rules" on page 7-21, "Explicit and Implicit Scope Terminators" on page 2-36, and "Delimited Scope Statements" on page 2-48.
2. If Format 1 is used, the values of the operands preceding the word TO are added together. The sum is added to the current value of identifier-2 and the result is stored immediately into identifier-2. This process is repeated respectively for each successive operand following the word TO in left-to-right order.
3. If Format 2 is used, the value of the operands preceding the word GIVING are added together. The sum is stored as the new value of each data item referenced by identifier-3, the resultant identifier.

-
4. If Format 3 is used, data items in identifier-1 are added to and stored in corresponding data items in identifier-2.
 5. The AIX VS COBOL system ensures that enough places are carried so as not to lose any significant digits during execution.

Example

The following example shows the ADD statement:

```
ADD TRAN-AMT TO TOTAL-AMT.  
ADD TRAN-AMT TO TOTAL-AMT  
    ON SIZE ERROR GO TO OVERFLOW-ERROR  
END-ADD.
```

ALTER Statement

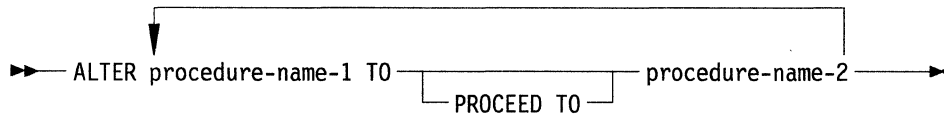
The ALTER statement is considered an obsolete COBOL language element although it is still supported.

Function

The ALTER statement modifies a predetermined sequence of operations.

General Format

The following figure shows the general format of the ALTER statement:



Syntax Rules

The following syntax rules apply to the ALTER statement:

1. Each procedure-name-1 is the name of a paragraph that contains a single sentence consisting of a GO TO statement without the DEPENDING phrase.
2. Each procedure-name-2 is the name of a paragraph or section in the Procedure Division.

General Rules

The following general rules apply to the ALTER statement:

1. Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1 so that subsequent executions of the modified GO TO statements cause transfer of control to procedure-name-2. Modified GO TO statements in independent segments may, under some circumstances, be returned to their initial states. Refer to "Independent Segments" on page 16-5.
2. A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

All other uses of the ALTER statement are valid and are performed even if procedure-name-1 is in an overlayable fixed segment.

Example

The following example shows the ALTER statement:

```
PROCEDURE DIVISION.
```

```
  :
```

```
  :
```

```
CHANGE-PATH.
```

```
  GO TO INITIALIZE-RTN.
```

```
INITIALIZE-RTN.
```

```
  DISPLAY "START PROGRAM".
```

```
  :
```

```
  :
```

```
  ALTER CHANGE-PATH TO REGULAR-PROCESS.
```

```
REGULAR-PROCESS.
```

```
  :
```

```
  :
```

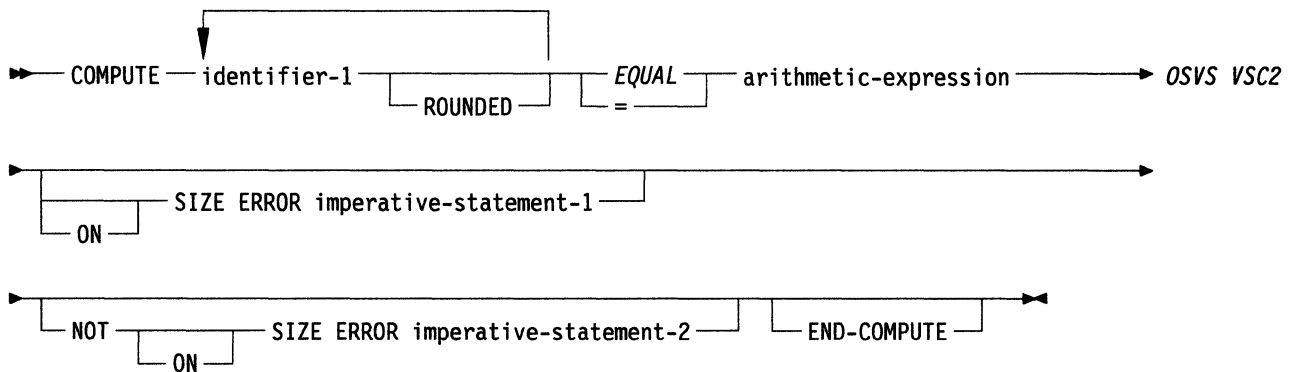
COMPUTE Statement

Function

The COMPUTE statement assigns to one or more data items the value of an arithmetic expression.

General Format

The following figure shows the general format of the COMPUTE statement:



Syntax Rule

Identifiers that appear only to the left of = must refer to either an elementary numeric item or an elementary numeric-edited item.

General Rules

The following syntax rules apply to the COMPUTE statement:

1. Refer to "ROUNDED Phrase" on page 7-19, "ON SIZE ERROR Phrase and NOT ON SIZE ERROR Phrase" on page 7-19, "Arithmetic Statement Rules" on page 7-20, "Overlapping Operand Rules" on page 7-21, "Multiple Results in Arithmetic Statement Rules" on page 7-21, "Explicit and Implicit Scope Terminators" on page 2-36, and "Delimited Scope Statements" on page 2-48.
2. An arithmetic expression consisting of a single identifier or literal provides a method of setting the value of identifier-1 equal to the value of the single identifier or literal.

-
3. If more than one identifier is specified for the result of the operation, that is preceding =, the value of the arithmetic expression is computed, and then this value is stored as the new value of each identifier-1 in turn.
 4. The COMPUTE statement allows the user to combine arithmetic operations without restrictions on the composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE.

Example

The following example shows the COMPUTE statement:

```
COMPUTE TOTAL = PRICE * UNITS * (1 + TAX-RATE)
      ON SIZE ERROR PERFORM SIZE-ERROR-RTN
      NOT ON SIZE ERROR PERFORM WRITE-LOG-FILE
END-COMPUTE.
```

CONTINUE Statement

Function

The CONTINUE statement is a no operation statement. It indicates that no executable statement is present.

General Format

The following example shows the general format of the CONTINUE statement:

▶▶— CONTINUE —◀◀

Syntax Rule

The CONTINUE statement may be used anywhere a conditional or an imperative statement may be used.

General Rule

The CONTINUE statement has no effect on the execution of the program.

Example

The following example shows the CONTINUE statement:

```
IF TOTAL-AMT < 1000
  CONTINUE
ELSE
  COMPUTE REBATE = TOTAL-AMT * 0.1
END-IF.
```

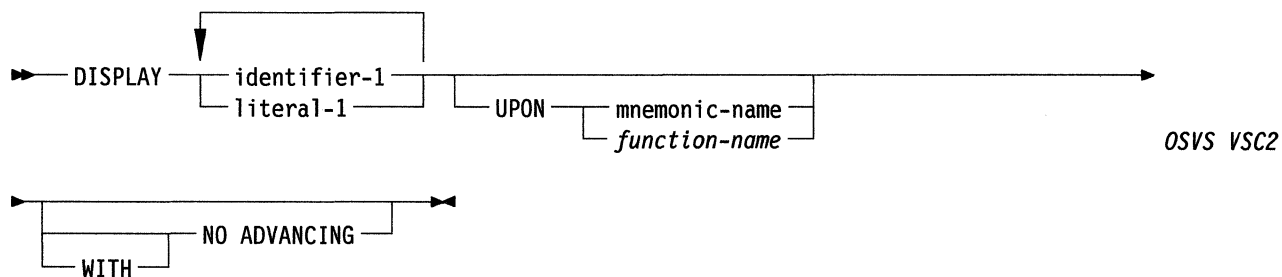
DISPLAY Statement

Function

The DISPLAY statement causes data to be transferred from specified data items to the console display screen. (See Chapter 18, "Screen-Handling" for additional formats of the DISPLAY statement.)

General Format

The following figure shows the general format of the DISPLAY statement:



Syntax Rules

The following syntax rules apply to the DISPLAY statement:

1. The mnemonic-name must be associated with a function-name in the SPECIAL-NAMES paragraph in the Environment Division. Refer to General Rule 9 on page 5-13 under "SPECIAL-NAMES Paragraph" for a list of valid function-names.
2. *Alternatively, function-name can itself be used instead of an associated OSVS VSC2 mnemonic name.*
3. Each literal may be any figurative constant, except ALL.
4. If the literal is numeric, it must be an unsigned integer.

General Rules

The following general rules apply to the DISPLAY statement:

1. When operands in a DISPLAY statement have USAGE other than DISPLAY, or have included signs, they are converted to USAGE DISPLAY with a separate sign. The size of the operand is taken as the size after conversion.
2. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.
3. *If the function-name COMMAND-LINE, or a mnemonic name associated with the function-name COMMAND LINE, is specified, then data overwrites the contents of a system-dependent command-line buffer. The contents may be retrieved by subsequent use of ACCEPT FROM COMMAND-LINE. Only one operand is allowed in this case.* **MF**

All other permissible function-names are treated as equivalent to CONSOLE and each operand is transferred to the console device in the order listed. The total size of data displayed is equal to the sum of the sizes of each operand minus the length of any trailing spaces in the last operand. The display starts from the current cursor position, overflowing onto the following line(s) if necessary. If the NO ADVANCING phrase is specified, the cursor is then left at the space following the last character displayed; otherwise it is positioned at the start of the next line. Scrolling may take place whenever the cursor is moved to a new line.

Example

The following example shows the DISPLAY statement:

```
DISPLAY ERROR-MSG UPON CONSOLE.  
DISPLAY "TOTAL AMOUNT: ", TOTAL-AMT.
```

DIVIDE Statement

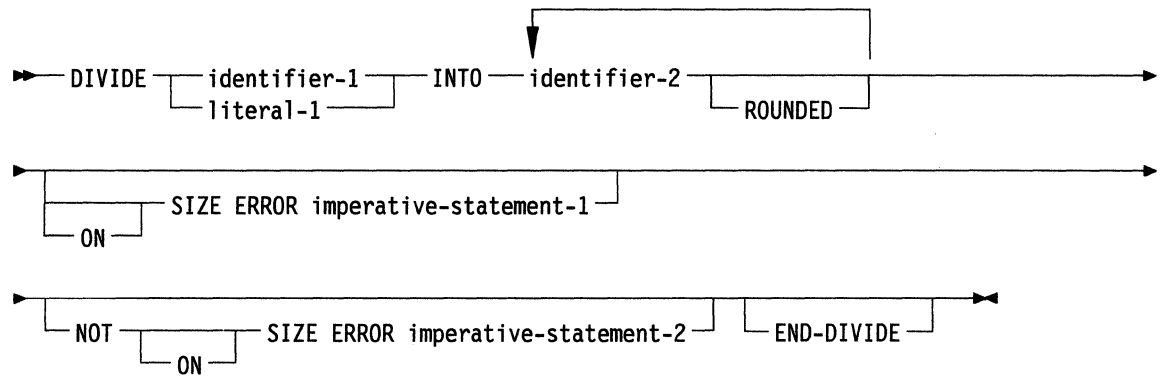
Function

The **DIVIDE** statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder.

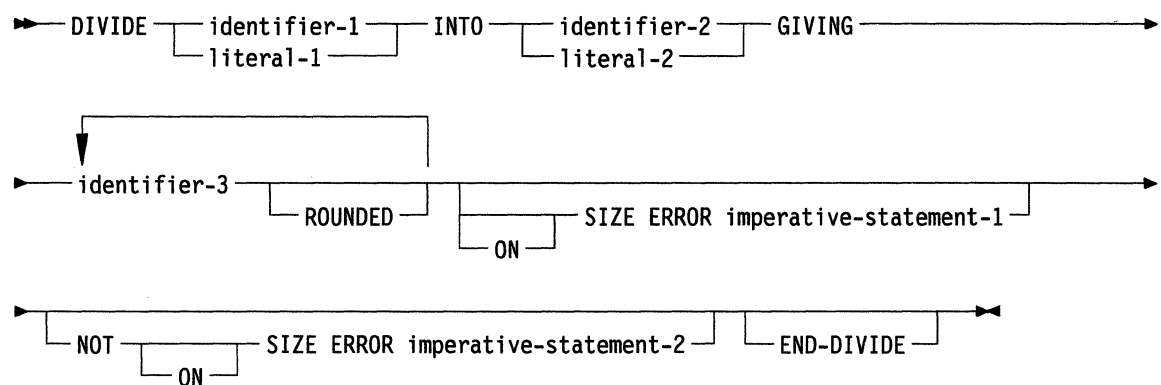
General Format

The following figures show the general format of the **DIVIDE** statement:

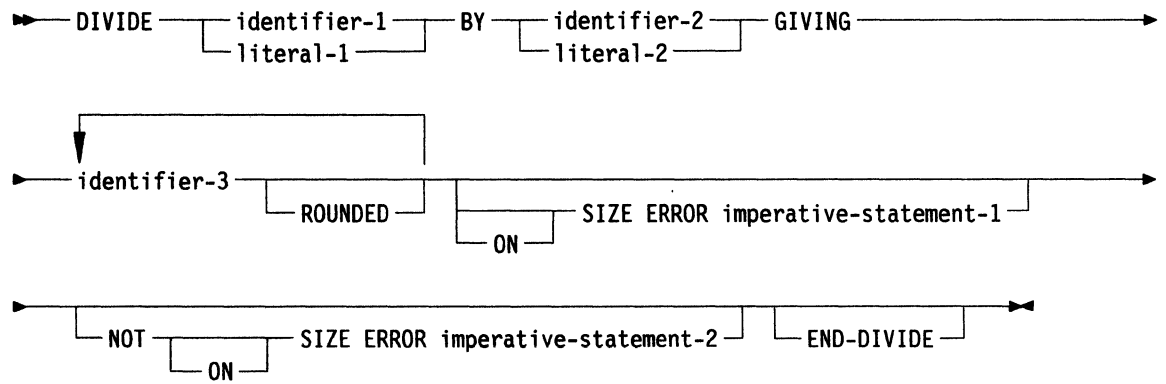
Format 1



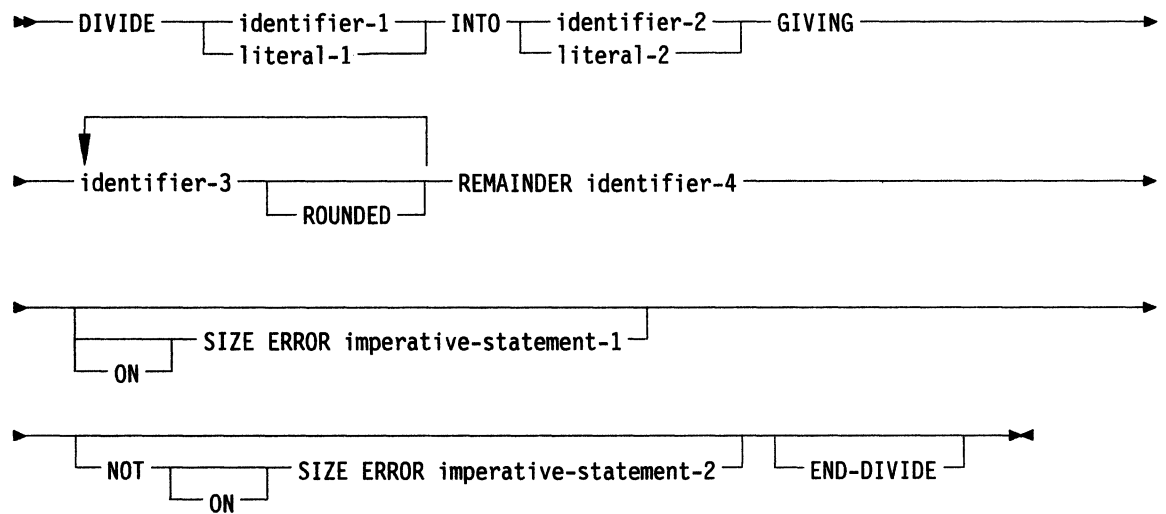
Format 2



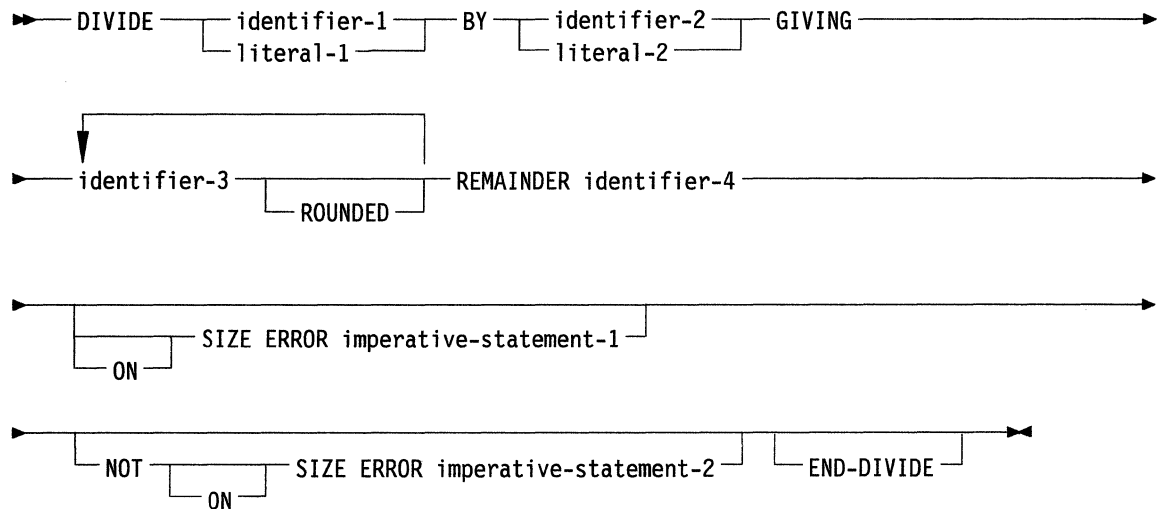
Format 3



Format 4



Format 5



Syntax Rules

The following syntax rules apply to the DIVIDE statement:

1. Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING or REMAINDER phrase must refer to either an elementary numeric item or an elementary numeric-edited item.
2. Each literal must be a numeric literal.
3. The composite of operands (the hypothetical data item resulting from the superimposition of all receiving data items, except the REMAINDER data item, of a given statement aligned on their decimal points) must not contain more than eighteen digits.

General Rules

The following general rules apply to the DIVIDE statement:

1. Refer to "ROUNDED Phrase" on page 7-19, "ON SIZE ERROR Phrase and NOT ON SIZE ERROR Phrase" on page 7-19, "Arithmetic Statement Rules" on page 7-20, "Overlapping Operand Rules" on page 7-21, "Multiple Results in Arithmetic Statement Rules" on page 7-21, "Explicit and Implicit Scope Terminators" on page 2-36, and "Delimited Scope Statements" on page 2-48. Also refer to general rules 5, 6, and 7 for a presentation of "ROUNDED Phrase" on page 7-19 and "ON SIZE ERROR Phrase and NOT ON SIZE ERROR Phrase" on page 7-19 as they pertain to Formats 4 and 5.
2. When Format 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by this quotient. The same holds true for any additional dividends.
3. When Format 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2 and the result is stored in identifier-3.
4. When Format 3 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2 and the result is stored in each data item referenced by identifier-3.

-
5. Formats 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. The remainder in COBOL is defined as the result of subtracting the product of the quotient (identifier-3) and the divisor from the dividend. If identifier-3 is defined as a numeric-edited item, the quotient used to calculate the remainder is an intermediate field which contains the unedited quotient. If **ROUNDED** is used, the quotient used to calculate the remainder is an intermediate field that contains the quotient of the **DIVIDE** statement, truncated rather than rounded.
 6. In Formats 4 and 5, the accuracy of the **REMAINDER** data item (identifier-4) is defined by the calculation described above. Appropriate decimal alignment truncation (not rounding) will be performed for the content of the data item referenced by identifier-4, as needed.
 7. When the **ON SIZE ERROR** phrase is used in Formats 4 and 5, the following rules apply:
 - a. If the size error occurs on the quotient, no remainder calculation is meaningful. Thus, the contents of the data items referenced by both identifier-3 and identifier-4 will remain unchanged.
 - b. If the size error occurs on the remainder, the contents of the data item referenced by identifier-4 remain unchanged. However, as with other instances of multiple results of arithmetic statements, the user will have to do his own analysis to recognize which situation has actually occurred.

Example

The following example shows the **DIVIDE** statement:

DIVIDE INCHES BY 12 GIVING FEET ROUNDED.

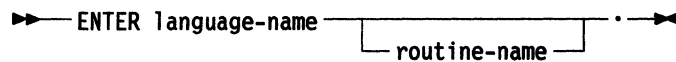
ENTER Statement

Function

The **ENTER** statement provides a means of allowing the use of more than one language in the same program.

General Format

The following figure shows the general format of the **ENTER** statement:



Syntax Rule

language-name and routine-name can be any user-defined word or alphanumeric literal.

General Rule

This statement is treated as if it were for documentation purposes only. Access to other languages can be achieved by means of **CALL**.

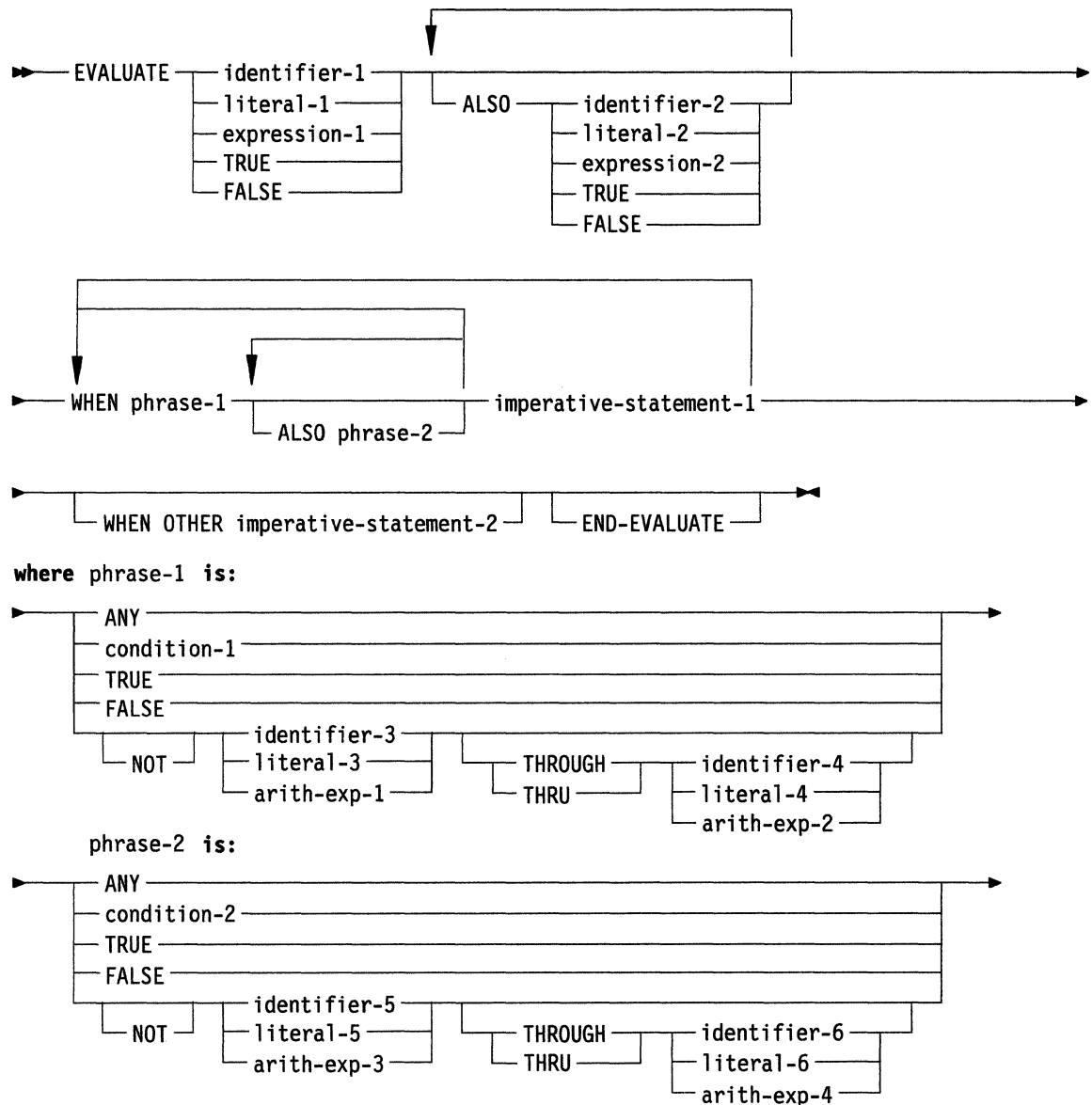
EVALUATE Statement

Function

The EVALUATE statement describes a multi-branch, multi-join structure. It can cause multiple conditions to be evaluated. The subsequent action of the object program depends on the results of these evaluations.

General Format

The following figure shows the general format of the EVALUATE statement:



Syntax Rules

The following syntax rules apply to the EVALUATE statement:

1. The operands or the words TRUE and FALSE (which appear before the first WHEN phrase of the EVALUATE statement) are referred to individually as selection subjects and collectively (for all those specified) as the set of selection subjects.
2. The operands or the words TRUE, FALSE, and ANY (which appear in a WHEN phrase of an EVALUATE statement) are referred to individually as selection objects and collectively (for all those specified in a single WHEN phrase) as the set of selection objects.
3. The words THROUGH and THRU are equivalent.
4. Two operands connected by a THROUGH phrase must be of the same class. The two operands thus connected constitute a single selection object.
5. The number of selection objects within each set of selection objects must be equal to the number of selection subjects.
6. 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:
 - a. 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. Refer to "Relation Condition" on page 7-9.
 - b. condition-1, condition-2, or the words TRUE or FALSE appearing as a selection object must correspond to a conditional expression or the words TRUE or FALSE in the set of selection subjects.
 - c. The word ANY may correspond to a selection subject of any type.
7. *A COBOL system directive (refer to the User's Guide) can be set to allow omission of the word ALSO. When this word is omitted, an operand starting with a unary sign must not immediately follow another operand in such a way that both operands, when considered together, form a single expression.* **MF**

DBCS Support

8. Where identifiers are permitted, they may reference DBCS data items. Where nonnumeric literals are permitted, DBCS literals are permitted.

End of DBCS Support

General Rules

The following general rules apply to the EVALUATE statement:

1. 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:
 - a. Any selection subject specified by identifier-1, identifier-2, and any selection object specified by identifier-3, identifier-5, without either the NOT or the THROUGH phrases, are assigned the value and class of the data item referenced by the identifier.

-
- b. Any selection subject specified by literal-1, literal-2, and any selection object specified by literal-3, literal-5, without either the NOT or the THROUGH phrases, are assigned the value and class of the specified literal. If literal-3, literal-5, is the figurative constant ZERO, it is assigned the class of the corresponding selection subject.
 - c. Any selection subject in which expression-1, expression-2, is specified as an arithmetic expression and any selection object, without either the NOT or the THROUGH phrases, in which arithmetic-expression-1, arithmetic-expression-3, is specified are assigned a numeric value according to the rules for evaluating an arithmetic expression. Refer to "Arithmetic Expressions" on page 7-7.
 - d. Any selection subject in which expression-1, expression-2 is specified as a conditional expression and any selection object in which condition-1, condition-2, is specified are assigned a truth value according to the rules for evaluating conditional expressions. Refer to "Conditional Expressions" on page 7-9.
 - e. 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.
 - f. Any selection object specified by the word ANY is not further evaluated.
 - g. If the THROUGH phrase is specified for a selection object, without the NOT phrase, the range of values is all values which, 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. Refer to "Relation Condition" on page 7-9. If the first operand is greater than the second operand, there are no values in the range.
 - h. If the NOT phrase is specified for a selection object, the values assigned to that item are not equal to the value, or range of values, which would have been assigned to the item had the NOT phrase not been specified.
2. The execution of the EVALUATE statement then proceeds as if the values assigned to the selection subjects and selection objects were compared to determine if any WHEN phrases satisfy the set of selection subjects. This comparison proceeds as follows:
 - a. Each selection object within the set of selection objects for the first WHEN phrase is compared to the selection subject with the same ordinal position within the set of selection subjects.
 - 1) 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 of 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. Refer to "Relation Condition" on page 7-9.
 - 2) If the items being compared are assigned truth values, the comparison is satisfied if the items are assigned the identical truth value.
 - 3) If the selection object being compared is specified by the word ANY, the comparison is satisfied, regardless of the value of the selection subject.
 - b. If the comparison is satisfied for every selection object in 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.
 - c. If the above comparison is not satisfied for one or more selection objects within the set of selection objects being compared, that set of selection objects does not satisfy the set of selection subjects.
 - d. The comparison 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.

-
3. After the comparison operation is completed, execution of the EVALUATE statement proceeds as follows:
 - a. If a WHEN phrase is selected, execution continues with the first imperative-statement-1 following the selected WHEN phrase.
 - b. If no WHEN phrase is selected and a WHEN OTHER phrase is specified, execution continues with imperative-statement-2.
 - c. The execution of the EVALUATE statement is terminated when execution reaches the end of the scope of the selected WHEN or WHEN OTHER phrase or when no WHEN phrase is selected and no WHEN OTHER phrase is specified. Refer to “Explicit and Implicit Scope Terminators” on page 2-36.

Example

The following example shows the EVALUATE statement:

```
EVALUATE GROSS > COST-1 + COST-2 + OVERHEAD
  WHEN TRUE PERFORM PROCESS-PROFIT
  WHEN FALSE PERFORM PROCESS-LOSS
END-EVALUATE.
```

EXAMINE Statement

Function

The EXAMINE statement replaces or counts the number of occurrences of a given character in a data item.

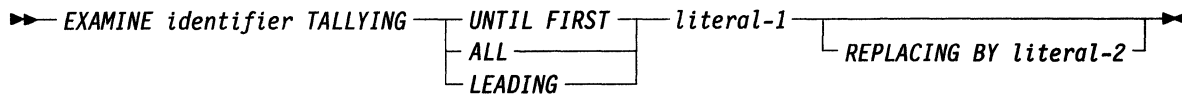
OSVS

General Format

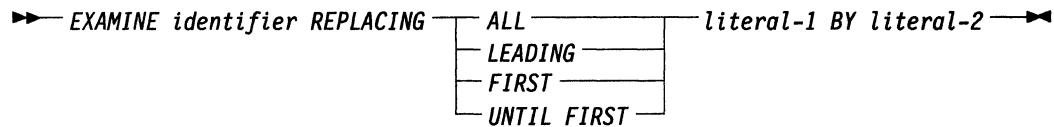
The following figures show the general format of the EXAMINE statement:

OSVS

Format 1



Format 2



Syntax Rules

The following syntax rules apply to the EXAMINE statement:

OSVS

- 1. The identifier must be described explicitly or implicitly as USAGE IS DISPLAY.*
- 2. Each literal must consist of a single character. If identifier is a numeric item, the literals must be either numeric literals, nonnumeric literals whose value is a single numeric digit, or the figurative constant ZERO. If identifier belongs to any other class, the literals may be numeric, non-numeric, or any figurative constant without the word ALL.*

General Rules

The following general rules apply to the EXAMINE statement:

OSVS

1. *Examination proceeds as follows:*
 - a. *For nonnumeric data items, examination starts at the leftmost character and proceeds to the right. Each character in the data item specified by the identifier is examined in turn.*
 - b. *If a data item referred to by the EXAMINE statement is numeric, it must consist of numeric characters and may possess an operational sign. Examination starts at the leftmost character (excluding the sign) and proceeds to the right. Each character except the sign is examined in turn. Regardless of where the sign is physically located, it is completely ignored by the EXAMINE statement.*
2. *The TALLYING option creates an integral count which replaces the value of a special register called TALLY. The count represents the number of:*
 - a. *Characters not equal to literal-1 encountered before the first occurrence of literal-1 when the UNTIL FIRST option is used.*
 - b. *Occurrences of literal-1 when the ALL option is used.*
 - c. *Occurrences of literal-1 prior to encountering a character other than literal-1 when the LEADING option is used.*
3. *When either of the REPLACING options is used, the replacement rules are as follows:*
 - a. *When the ALL option is used, literal-2 is substituted for each occurrence of literal-1.*
 - b. *When the LEADING option is used, the substitution of literal-2 terminates as soon as a character other than literal-1 is encountered. The substitution of literal-2 also terminates when the right-hand boundary of the data item is encountered.*
 - c. *When the FIRST option is used, the first occurrence of literal-1 is replaced by literal-2.*
 - d. *When the UNTIL FIRST option is used, the substitution of literal-2 terminates as soon as literal-1 or the right-hand boundary of the data item is encountered.*

Example

The following example shows the EXAMINE statement changing the CURR-DATE format from mm/dd/yy to mm-dd-yy.

```
EXAMINE CURR-DATE REPLACING ALL '/' BY '-'
```

EXEC(UTE) Statement

Function

The EXEC(UTE) statement is provided as a linkage mechanism to allow control to be passed to non-COBOL subsystems. MF

General Format

The following figure shows the general format of the EXEC(UTE) statement: MF

▶— EXEC — text-name — text-data END-EXEC —▶

Syntax Rule

text-data may be any textual data not containing the string END-EXEC imbedded within it. MF

General Rule

The statement is compiled as a CALL text-name USING text-data-buffer statement (refer to "CALL Statement" on page 11-36) where text-data-buffer contains all the text-data from the EXEC statement (space compressed) for further parsing by the CALLED program text-name. MF

EXHIBIT Statement

Function

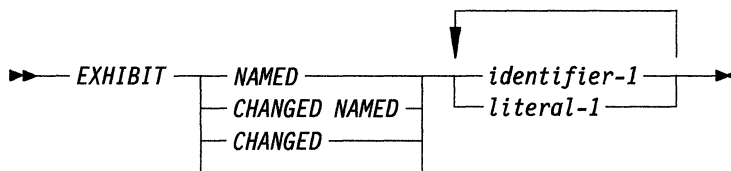
The *EXHIBIT* statement causes a display (optionally conditional) of the literals, and/or identifiers (optionally preceded by the identifier name) specified in the statement.

OSVS MF

General Format

The following figure shows the general format of the *EXHIBIT* statement:

OSVS MF



Syntax Rules

The following syntax rules apply to the *EXHIBIT* statement:

OSVS MF

1. Identifiers specified in the *EXHIBIT* statement may be of any class of data. *TALLY* and *RETURN-CODE* are the only special registers that can be used as identifiers.
2. *CHANGED* and *NAMED* may both be omitted. The effect is as if *CHANGED* were specified. **If the *IBM-MS* option is given, then the effect is as if *NAMED* were specified.**

IBM-MS

General Rules

OSVS

The following general rules apply to the EXHIBIT statement:

1. *Literals and identifiers displayed by the EXHIBIT statement are separated by a space on the displayed line.*
2. *Each literal may be any figurative constant other than ALL.*
3. *If the literal is numeric, it must be an unsigned integer.*
4. *Each execution of an EXHIBIT NAMED statement displays each identifier or literal specified, with each identifier (including any qualifiers and subscripts) followed by an = (equal sign) and its current value. They all appear on a single line in the order in which they appear in the statement.*
5. *Each execution of an EXHIBIT CHANGED NAMED statement displays each identifier or literal specified with each identifier (including any qualifiers and subscripts) followed by an = and its current value. They all appear on a single line in the order in which they appear in the statement. However, the display for each identifier (name and value) is conditional on the value of that identifier having changed since the last execution of the current EXHIBIT statement. If one or more of the identifier values have not changed, neither the name nor the value is printed for those identifiers. If none of the identifier values have changed, and there are no literals specified, there is no display (display of a blank line is suppressed).*
6. *Each execution of an EXHIBIT CHANGED statement displays the current value of each identifier or literal. They all appear on a single line in the order in which they appear in the statement. However, the value display for each identifier is conditional on the value of that identifier having changed since the last execution of the current EXHIBIT statement. If one or more of the identifier values have not changed, the value for those identifiers is not printed; spaces are inserted instead. If none of the identifier values have changed, and no literals are specified, a blank line is displayed (display of a blank line is not suppressed).*

Example

The following example shows the EXHIBIT statement:

```
EXHIBIT NAMED P-ID P-PHONE P-COUNT.
```

EXIT Statement

Function

The EXIT statement provides a common end point for a series of procedures. Refer to Chapter 11, “Interprogram Communication” for an additional format of this statement.

General Format

The following figure shows the general format of the EXIT statement:

▶— EXIT —▶

Syntax Rules

The following syntax rules apply to the EXIT statement:

1. The EXIT statement should appear in a sentence by itself. *This rule is not enforced.* MF
2. The EXIT sentence should be the only sentence in the paragraph. *This rule is not enforced.* MF

General Rule

An EXIT statement serves only to enable the user to assign a procedure-name to a given point in a program. Such an EXIT statement has no other effect on the compilation or execution of the program.

Example

The following example shows the EXIT statement:

```
MAIN-PROG.  
  :  
  :  
  PERFORM INVENTORY THRU INV-EXIT.  
  :  
  :  
INVENTORY.  
  :  
  :  
INV-1.  
  :  
  :  
INV-2.  
  :  
  :  
INV-EXIT.  
  EXIT.
```

GO TO Statement

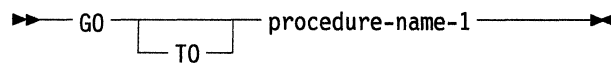
Function

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

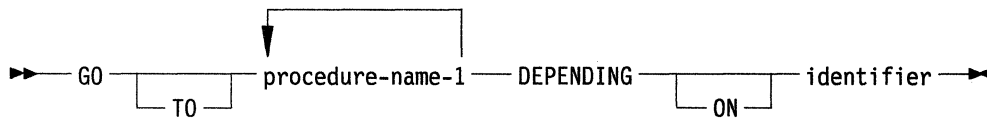
General Format

The following figures show the general format for the GO TO statement:

Format 1



Format 2



Syntax Rules

The following syntax rules apply to the GO TO statement:

1. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.
2. When a paragraph is referenced by an ALTER statement, that paragraph can consist only of a paragraph header followed by a Format 1 GO TO statement.
3. A Format 1 GO TO statement, without procedure-name-1, can only appear in a single statement paragraph.
4. If a GO TO statement represented by Format 1 appears in a consecutive sequence of imperative statements without a sentence, it appears as the last statement in that sequence.
5. In Format 2, it is recommended that at least two procedure names should be specified. However, a Format 2 statement may be written with only one procedure name.

General Rules

The following general rules apply to the GO TO statement:

1. When a GO TO statement represented by Format 1 is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement has been modified by an ALTER statement.
2. If procedure-name-1 is not specified in Format 1, an ALTER statement referring to this GO TO statement must be executed prior to the execution of this GO TO statement.
3. When a GO TO statement represented by Format 2 is executed, control is transferred to the corresponding procedure-name-1 depending on the value of the identifier being 1, 2, ..., n. If the value of the identifier is anything other than the positive or unsigned integers 1, 2, ..., n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

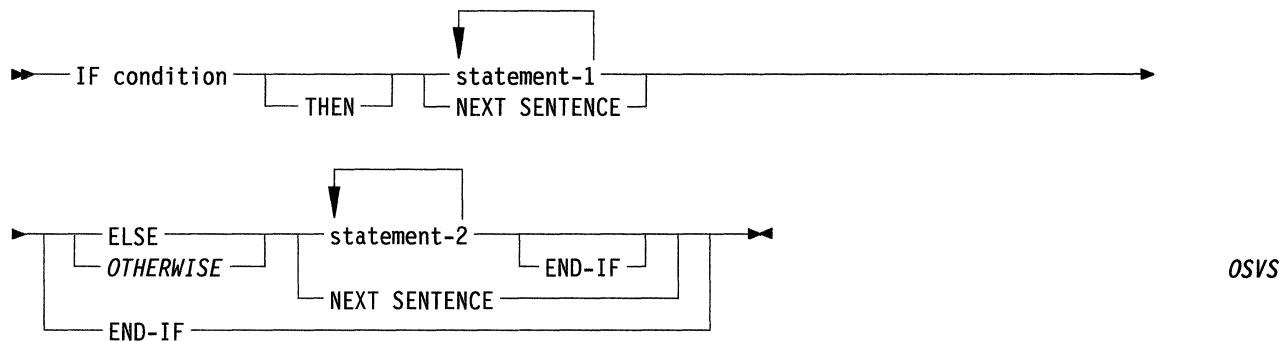
IF Statement

Function

The IF statement causes a condition to be evaluated (refer to “Conditional Expressions” on page 7-9). The subsequent action of the object program depends on whether the value of the condition is true or false.

General Format

The following figure shows the general format of the IF statement:



Syntax Rules

The following syntax rules apply to the IF statement:

1. statement-1 and statement-2 represent either an imperative statement or a conditional statement and either may be followed by a conditional statement.
2. If the END-IF phrase is specified, the NEXT SENTENCE phrase must not be specified.

General Rules

The following general rules apply to the IF statement:

1. The scope of an IF statement may be terminated by any of the following:
 - a. An END-IF phrase at the same level of nesting.
 - b. A separator period.
 - c. If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting.
2. When an IF statement is executed, the following transfers of control occur:
 - a. If the condition is true, statement-1 is executed if specified. If statement-1 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-1 does not contain a procedure branching or conditional statement, the ELSE phrase, if specified, is ignored and control passes to the end of the IF statement.

-
- b. If the condition is true and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.
 - c. If the condition is false, statement-1 or its surrogate NEXT SENTENCE is ignored, and statement-2, if specified, is executed. If statement-2 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-2 does not contain a procedure branching or conditional statement, control passes to the end of the IF statement. If the ELSE statement-2 phrase is not specified, statement-1 is ignored and control passes to the end of the IF statement.
 - d. If the condition is false, and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.
3. statement-1 and/or statement-2 may contain an IF statement. In this case the IF statement is said to be nested.

IF statements within IF statements may be considered as paired IF ELSE and END-IF combinations, proceeding from left to right. Any END-IF encountered is considered to apply to the immediately preceding IF that has not been already paired with an END-IF. Any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE or END-IF.

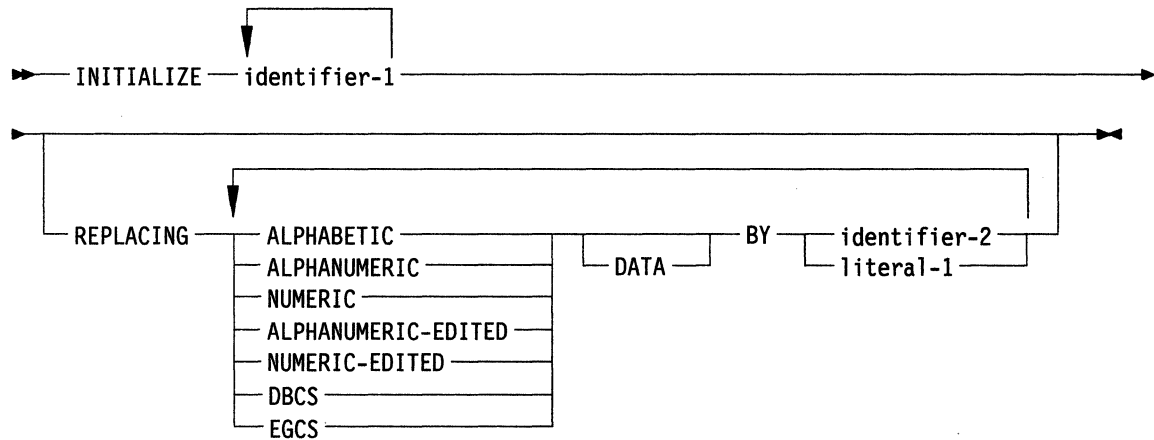
INITIALIZE Statement

Function

The INITIALIZE statement provides the ability to set selected types of data fields to predetermined values; for example, numeric data to zeros or alphanumeric data to spaces.

General Format

The following figure shows the general format of the INITIALIZE statement:



Syntax Rules

The following syntax rules apply to the INITIALIZE statement:

1. literal-1 and the data item referenced by identifier-2 represent the sending area; the data item referenced by identifier-1 represents the receiving area.
2. Each category stated in the REPLACING phrase must be a permissible category as a receiving operand in a MOVE statement, where the corresponding data item referenced by identifier-2 or literal-1 is used as the sending operand. Refer to "MOVE Statement" on page 7-65.
3. The same category cannot be repeated in a REPLACING phrase.
4. The description of the data item referenced by identifier-1 or any items subordinate to identifier-1 may not contain the DEPENDING phrase of the OCCURS clause.
5. An index data item may not appear as an operand of an INITIALIZE statement.
6. The data description entry for the data item referenced by identifier-1 must not contain a RENAME clause.

General Rules

The following general rules apply to the INITIALIZE statement:

1. The key word following the word REPLACING corresponds to a category of data defined in "Class Condition" on page 7-13.
2. Whether identifier-1 references an elementary item or a group item, all operations are performed as if a series of MOVE statements had been written, each of which has an elementary item as its receiving field, subject to the following rules:

If the REPLACING phrase is specified:

- a. 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.
- b. 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 follows: 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; the only exceptions are those fields specified in 3.

3. Index data items and elementary FILLER data items are not affected by the execution of an INITIALIZE statement.
4. Any item that is subordinate to a receiving area identifier and contains the REDEFINES clause or any item that is subordinate to such an item is excluded from this operation. However, a receiving area identifier may itself have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause.
5. When the statement is written without the REPLACING phrase, data items of the categories alphabetic, alphanumeric, and alphanumeric-edited are set to spaces; data items of the categories numeric and numeric-edited are set to zeros. In this case, the operation is as if each affected data item is the receiving area in an elementary MOVE statement with the indicated source literal (that is, spaces or zeros).
6. In all cases, the content of the data item referenced by identifier-1 is set to the indicated value in the order (left to right) of the appearance of identifier-1 in the INITIALIZE statement. Within this sequence, where identifier-1 references a group item, affected elementary items are initialized in the sequence of their definition within the group.
7. If identifier-1 occupies the same storage area as identifier-2, the result of the execution of this statement is undefined even if they are defined by the same description entry. Refer to "Overlapping Operand Rules" on page 7-21.

DBCS Support

8. A DBCS data item (USAGE DISPLAY-1) or literal (PIC G) may be used anywhere an identifier or literal is specified. The category stated in the REPLACING phrase may be DBCS or EGCS.

End of DBCS Support

Example

The following example shows the INITIALIZE statement:

```
INITIALIZE INVENTORY-RECORD
           SALES-RECORD
           TOTAL-UNITS
           UNIT-PRICE
           TOTAL-AMOUNT
REPLACING ALPHABETIC DATA BY SPACES
          ALPHANUMERIC DATA BY SPACES
          NUMERIC DATA BY ZEROS.
```

INSPECT Statement

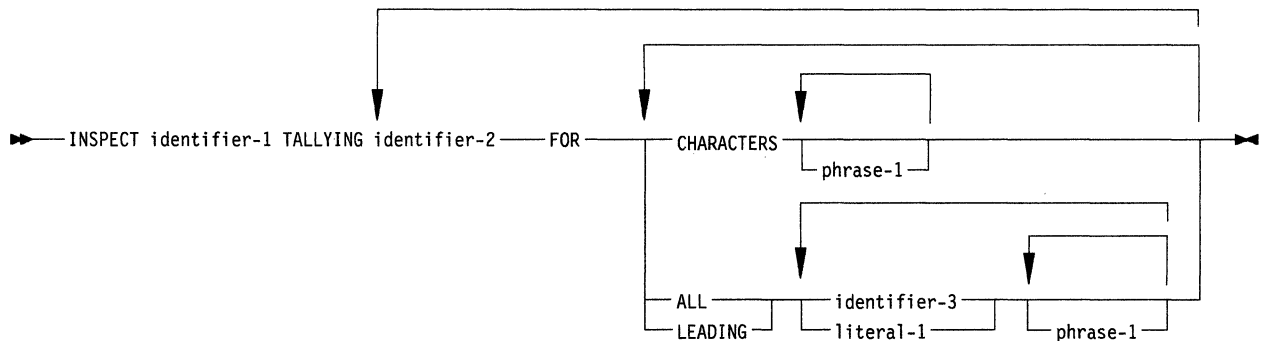
Function

The INSPECT statement provides the ability to tally (Format 1), replace (Format 2), tally and replace (Format 3), or convert (Format 4) occurrences of single characters or group of characters in a data item.

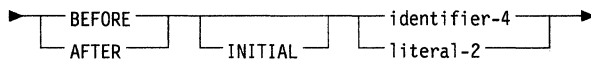
General Format

The following figures show the general format of the INSPECT statement:

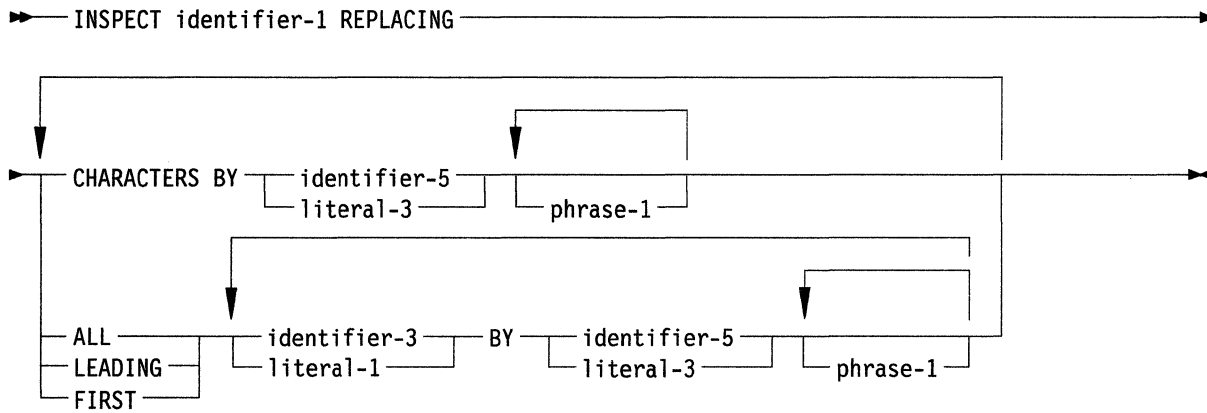
Format 1



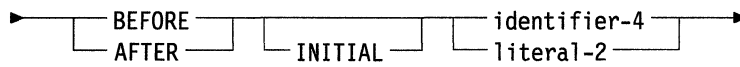
where phrase-1 is:



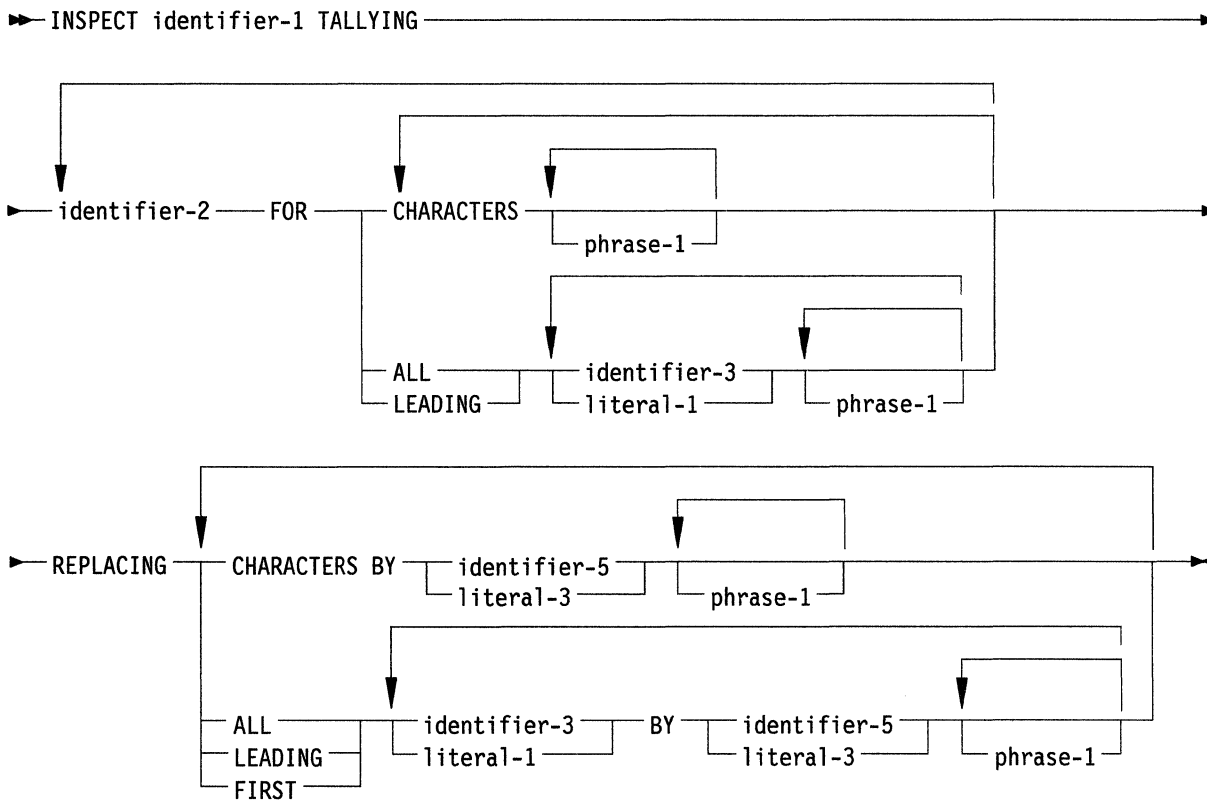
Format 2



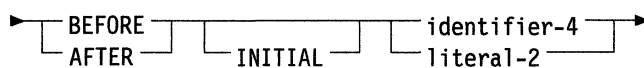
where phrase-1 is:



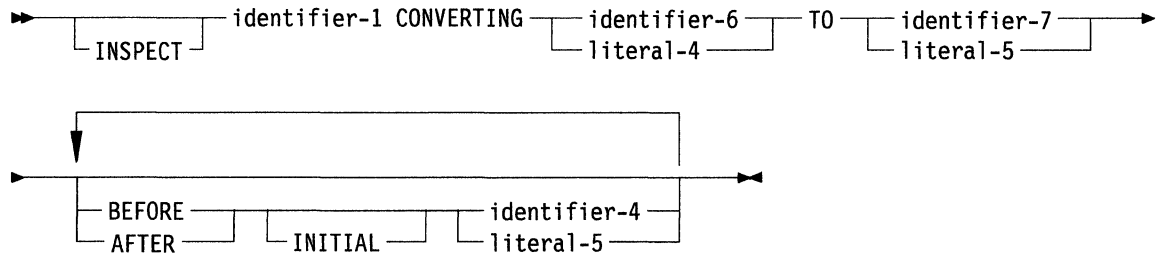
Format 3



where phrase-1 is:



Format 4



Syntax Rules

The following syntax rules apply to the INSPECT statement:

All Formats

1. identifier-1 must reference either a group item or any category of elementary item, described (either implicitly or explicitly) as USAGE IS DISPLAY.
2. identifier-3 ... identifier-n must reference either an elementary alphabetic, alphanumeric or numeric item described (either implicitly or explicitly) as USAGE IS DISPLAY.
3. Each literal must be nonnumeric and may be any figurative constant, except ALL. If literal-1, literal-2, or literal-4 is a figurative constant, it refers to an implicit one character data item.
4. No more than one BEFORE phrase and/or one AFTER phrase can be specified for any one ALL, LEADING, CHARACTERS, FIRST, or CONVERTING phrase.
5. literal-1, literal-2, literal-3, literal-4, and literal-5, and the data items referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, can be any number of characters in length up to the limit allowed for literals or data items.

DBCS Support

6. All identifiers and literals except the count field (identifier-2) must be DBCS items, either USAGE DISPLAY-1 or DBCS literals, if any are DBCS items. identifier-2 cannot be a DBCS item. DBCS characters, not bytes of data, are tallied in identifier-2.

End of DBCS Support

Formats 1 and 3 Only

7. identifier-2 must reference an elementary numeric data item.

Formats 2 and 3 Only

8. The size of the data referenced by literal-3 or identifier-5 must be equal to the size of the data referenced by literal-1 or identifier-3. When a figurative constant is used as literal-3, the size of the figurative constant is equal to the size of literal-1 or the size of the data item referenced by identifier-3.
9. When the CHARACTERS phrase is used, literal-2, literal-3, or the size of the data item referenced by identifier-4, identifier-5 must be one character in length.

Format 4 Only

10. The size of literal-5 or the data item referenced by identifier-7 must be equal to the size of literal-4 or the data item referenced by identifier-6. When a figurative constant is

used as literal-5, the size of the figurative constant is equal to the size of literal-4 or the size of the data item referenced by identifier-6.

11. The same character must not appear more than once either in literal-4 or in the data item referenced by identifier-6.

General Rules

The following general rules apply to the INSPECT statement:

All Formats

1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position as described in general rules 5, 6, and 7.
2. For use in the INSPECT statement, the contents of the data item referenced by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 will be treated as follows:
 - a. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 is described as alphanumeric, the INSPECT statement treats the contents of each such identifier as a character string.
 - b. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 is described as alphanumeric-edited, numeric-edited or unsigned numeric, the data item is inspected as though it had been redefined as alphanumeric (see 2a) and the INSPECT statement had been written to reference the redefined data item.
 - c. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 is described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item with length equal to the length of the signed item excluding any separate sign position and the rules in 2b applied. Refer to "MOVE Statement" on page 7-65.
3. In general rules 4 through 11 on page 7-62, all references to literal-1, literal-2, literal-3, literal-4, and literal-5 apply equally to the contents of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, respectively.

Formats 1 and 2

4. During inspection of the contents of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (Format 1) or replaced by literal-3 (Format 2).
5. The comparison operation to determine the occurrences of literal-1 to be tallied or to be replaced, occurs as follows:
 - a. The operands of the TALLYING or REPLACING phrases are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1 is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by identifier-1. literal-1 matches that portion of the contents of the data item referenced by identifier-1 if, they are equal, character for character, and:
 - 1) If neither LEADING nor FIRST is specified; or
 - 2) If the LEADING adjective applies to literal-1 and literal-1 is a leading occurrence as defined in rule 9 on page 7-62 and rule 12 on page 7-62; or
 - 3) If the FIRST adjective applies to literal-1 and literal-1 is the first occurrence as defined in rule 8 on page 7-61 and rule 11 on page 7-62.
 - b. If no match occurs in the comparison of the first literal-1, the comparison is repeated with each successive literal-1, until either a match is found or there is no successive literal-1. When there is no successive literal-1, the character position in

the data item referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1.

- c. Whenever a match occurs, tallying and/or replacing takes place as described in rule 8 through rule 10 on page 7-62. The character position in the data item referenced by identifier-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with the first literal-1.
 - d. The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.
 - e. If the CHARACTERS phrase is specified, an implied one-character operand participates in the cycle described in rule 5a on page 7-60 through rule 5d, except that no comparison to the contents of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the contents of the data item referenced by identifier-1 participating in the current comparison cycle.
6. The comparison operation defined in rule 5 on page 7-60 is affected by the BEFORE and AFTER phrases as follows:
- a. If neither the BEFORE nor the AFTER phrase is specified, literal-1 or the implied operand of the CHARACTERS phrase is first eligible to participate in matching at the leftmost character position of identifier-1.
 - b. If the BEFORE phrase is specified, the associated literal-1, or the implied operand of the CHARACTERS phrase participates only in those comparison cycles that involve that portion of the data item content referenced by identifier-1 from its leftmost character position up to, but not including, the first occurrence of literal-2 within the contents of the data item referenced by literal-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in rule 5 on page 7-60 is begun. If, on any comparison cycle, literal-1, or the implied operand of the CHARACTERS phrase, is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, within the contents of the data item referenced by identifier-1, its associated literal-1, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.
 - c. If the AFTER phrase is specified, the associated literal-1, or the implied operand of the CHARACTERS phrase, may participate only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1. The area affected is from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, within the contents of the data item referenced by identifier-1, to the rightmost character position of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in rule 5 on page 7-60 is begun.

If, on any comparison cycle, literal-1, or the implied operand of the CHARACTERS phrase, is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2 within the contents of the data item referenced by identifier-1, its associated literal-1, or the implied operand of the CHARACTERS phrase, is never eligible to participate in the comparison operation.

Format 1

7. The contents of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.
8. The rules for tallying are as follows:

-
- a. If the ALL phrase is specified, the contents of the data item referenced by identifier-2 are incremented by one for each occurrence of literal-1 matched within the contents of the data item referenced by identifier-1.
 - b. If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 are incremented by one for the first and each subsequent contiguous occurrence of literal-1. This is matched within the contents of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.
 - c. If the CHARACTERS phrase is specified, the contents of the data item referenced by identifier-2 are incremented by one for each character matched, in respect to Formats 1 and 2, rule 5e on page 7-61, within the contents of the data item referenced by identifier-1.
9. If identifier-1, identifier-3, or identifier-4 occupies the same storage area as identifier-2, the result of the execution of this statement is undefined, even if the identifiers are defined by the same data description entry. Refer to "Overlapping Operand Rules" on page 7-21.

Format 2

10. The required words ALL, LEADING, and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.
11. The rules for replacement are as follows:
 - a. When the CHARACTERS phrase is specified, each character matched in respect to Format 1 and 2 item 5e on page 7-61 in the contents of the data item referenced by identifier-1, is replaced by literal-3.
 - b. When the adjective ALL is specified, each occurrence of literal-1 matched in the contents of the data item referenced by identifier-1 is replaced by literal-3.
 - c. When the adjective LEADING is specified, the first and each subsequent contiguous occurrence of literal-1 matched in the contents of the data item referenced by identifier-1 is replaced by literal-3. This replacement transpires provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.
 - d. When the adjective FIRST is specified, the leftmost occurrence of literal-1 matched within the contents of the data item referenced by identifier-1 is replaced by literal-3.
12. If identifier-3, identifier-4, or identifier-5 occupies the same storage area as identifier-1, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. Refer to "Overlapping Operand Rules" on page 7-21.

Format 3

13. A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement. The other statement is interpreted as being a Format 2 statement with REPLACING phrases identical to those specified in the Format 3 statement. The general rules given for matching and counting apply to the Format 1 statement and the general rules given for matching and replacing apply to the Format 2 statement. Refer to Figure 7-2 on page 7-63.

Format 4

14. A Format 4 INSPECT statement is interpreted and executed as though a Format 2 INSPECT statement specifying the same identifier-1 had been written with a series of ALL phrases, one for each character of literal-4. The effect is as if each of these ALL phrases referenced, as literal-1, a single character of literal-4 and referenced, as literal-3, the corresponding single character of literal-5. Correspondence between the characters of literal-4 and the characters of literal-5 is by ordinal position within the data item.

15. 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.

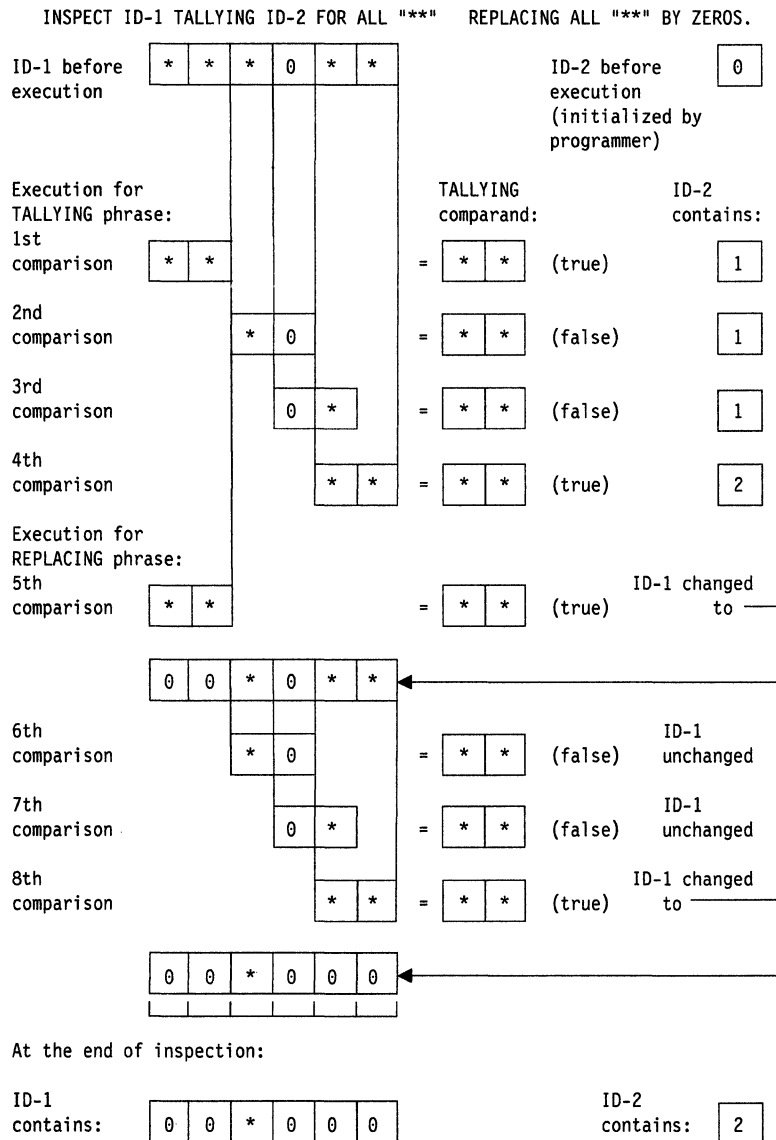


Figure 7-2. INSPECT Statement and the Execution Result

Examples

Six examples of the use of the INSPECT statement follow. In each case, it is assumed that the count fields have value zero before the statement is executed.

1. INSPECT word TALLYING count FOR LEADING "L" BEFORE INITIAL "A" count-1 FOR LEADING "A" BEFORE INITIAL "L".

Where word = LARGE, count = 1, count-1 = 0.

Where word = ANALYST, count = 0, count-1 = 1.

2. INSPECT word TALLYING count FOR ALL "L" REPLACING LEADING "A" BY "E" AFTER INITIAL "L".

Where word = CALLAR, count = 2, word = CALLAR.

Where word = SALAMI, count = 1, word = SALEMI.

Where word = LATTER, count = 1, word = LETTER.

3. INSPECT word REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

Where word = ARXAX, word = GRXAX.

Where word = HANDAX, word = HGNDGX.

4. INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL "J" REPLACING ALL "A" BY "B".

Where word = ADJECTIVE, count = 6, word = BJECTIVE.

Where word = JACK, count = 3, word = JBCK.

Where word = JUJMAB, count = 5, word = JUJMAB.

5. INSPECT word REPLACING ALL "X" BY "Y", "B" BY "Z", "W" BY "Q" AFTER INITIAL "R".

Where word = RXXBQWY, word = RYYZQQY.

Where word = YZACDWR, word = YZACDWZR.

Where word = RAWRXEB, word = RAQRYEZ.

6. INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".

word before: 1 2 X Z A B C D

word after: B B B B B A B C D

MOVE Statement

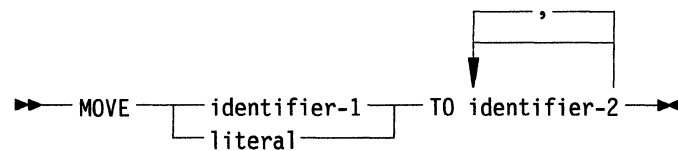
Function

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

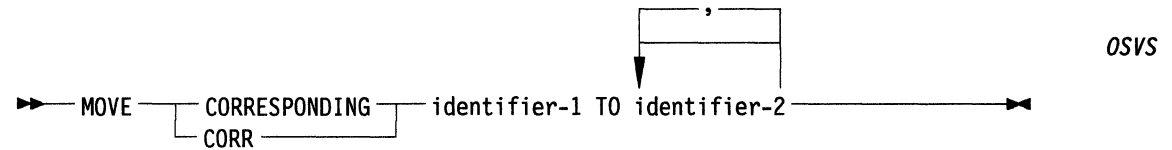
General Format

The following figures show the general format of the MOVE statement:

Format 1



Format 2



Syntax Rules

The following syntax rules apply to the MOVE statement:

1. identifier-1 and literal represent the sending area; identifier-2 represents the receiving area.
2. CORR is an abbreviation for CORRESPONDING.
3. When the CORRESPONDING phrase is used, all identifiers must be group items.
4. An index data item cannot appear as an operand of a MOVE statement. Refer to "USAGE Clause" on page 6-39.

General Rules

The following general rules apply to the MOVE statement:

1. If the CORRESPONDING phrase is used, selected items within identifier-1 are moved to selected items within identifier-2, according to the rules given in "CORRESPONDING Phrase" on page 7-20. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements. *The process is repeated for each destination group.* OSVS
2. The data designated by the literal or identifier-1 is moved first to identifier-2, then to identifier-3, The rules referring to identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with identifier-2, ..., is evaluated immediately before the data is moved to the respective data item.

Any subscripting or indexing associated with identifier-1 is evaluated only once, which is immediately before data is moved to the first of the receiving operands. The result of the statement:

```
MOVE a(b) TO b, c(b)
```

is equivalent to:

```
MOVE a(b) TO temp
```

```
MOVE temp TO b
```

```
MOVE temp TO c(b)
```

where temp is an intermediate result item provided by the AIX VS COBOL system.

3. Any MOVE in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric-edited, alphanumeric-edited. These categories are described in "PICTURE Clause" on page 6-18. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

The following rules apply to an elementary move to these data items.:

- a. The figurative constant SPACE, or an alphanumeric-edited, or alphabetic data item must not be moved to a numeric or numeric-edited data item.
 - b. A numeric-edited data item must not be moved to a numeric-edited data item.
 - c. A numeric literal, the figurative constant ZERO, a numeric data item, or a numeric-edited data item must not be moved to an alphabetic data item.
 - d. All other elementary moves are legal and are performed according to the rules given in rule 4.
4. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:
 - a. When an alphanumeric-edited or alphanumeric item is a receiving item, alignment and any necessary space filling takes place as defined in "Standard Alignment Rules" on page 2-19. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign will not be moved; if the operational sign occupies a separate character position (refer to "SIGN Clause" on page 6-35), that character will not be moved and the size of the sending item will be considered to be one less than its actual size (in terms of standard format characters).

-
- b. When a numeric or numeric-edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as defined in "Standard Alignment Rules" on page 2-19, except where zeros are replaced because of editing requirements.

When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. Refer to "SIGN Clause" on page 6-35. Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

When a numeric-edited item is the sending item, and the receiving item is numeric, the sending item has any nonnumeric characters removed; any sign or decimal point is moved, along with the digits, to form a true numeric item. All other characters are discarded.

- c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined in "Standard Alignment Rules" on page 2-19. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

- d. *If the sending item is a pointer item, no conversion takes place. The receiving item must also be a pointer item.* VSC2

- e. *When a noninteger numeric item is moved to an alphanumeric item:* MF

1) *The move is alphanumeric, that is, it is performed from left to right with any necessary space-filling or truncation taking place on the right.*

2) *Any decimal point position is ignored. Decimal point character or position is not needed for it appears in the receiving item.*

3) *Digit positions represented in the source item by Ps are ignored. No characters or character positions representing them appear in the receiving item.*

4) *Zeros appearing explicitly in the same item are treated as nonzero digits.*

DBCS Support

- f. DBCS moves include DBCS data items (defined as USAGE DISPLAY-1) and DBCS literals. 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.

End of DBCS Support

5. Any move that is not an elementary move is treated exactly as if it were an alphanumeric-to-alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in 4 on page 12-8.

6. Table 7-3 on page 7-68 summarizes the legality of the various types of MOVE statements. The general rule reference indicates the rule that prohibits the move or the behavior of a legal move.

DBCS Support

7. If either the sending or receiving item is a DBCS (USAGE DISPLAY-1) item, then both must be DBCS items. No data conversion is done; however, the data will be either truncated or padded with DBCS spaces on the right, as necessary. The figurative constant SPACE/SPACES can be the DBCS sending item.
8. No conversion of characters moved to or from a PIC G data item occurs, which means that the field may become corrupted. We recommend that you avoid MOVE statements involving PIC G fields, unless both source and target are defined as PIC G. If you move data from a PIC G data item to a PIC G EDITED data item, the relevant space characters are inserted into the target item.

End of DBCS Support

Table 7-3. MOVE Statement Data Categories

Category of Sending Item	Category of Receiving Data Item*					
	Alpha- betic	Alpha numeric- Edited Alpha- numeric	Numeric integer Numeric Non- Integer	Numeric- Edited	Pointer	DBCS ¹
Alphabetic	Yes/4c	Yes/4a	No/3a	No/3a	No/4d	No/4f
Alphanumeric	Yes/4c	Yes/4a	Yes/4b	Yes/4b	No/4d	No/4f
Alphanumeric- Edited	Yes/4c	Yes/4a	No/3a	No/3a	No/4d	No/4f
Numeric Integer Noninteger	No/2b No/3b	Yes/4a Yes/4e MF	Yes/4b Yes/4b	Yes/4b Yes/4b	No/4d No/4d	No/4f No/4f
Numeric-Edited	No/3b	Yes/4a	Yes/4b MF	No/3b	No/4d	No/4f
Pointer	No/4d	No/4d	No/4d	No/4d	Yes/4d	No/4f
DBCS ²	No/4f	No/4f	No/4f	No/4f	No/4f	Yes/4f
Note:						
* The relevant rule number is quoted in these columns.						
¹ Includes DBCS data items (USAGE DISPLAY-1).						
² Includes both DBCS data items (USAGE DISPLAY-1) and DBCS literals.						

Example

The following example shows the MOVE statement:

```
MOVE 0 TO COUNT.
MOVE CORR IN-REC TO OUT-REC.
```

MULTIPLY Statement

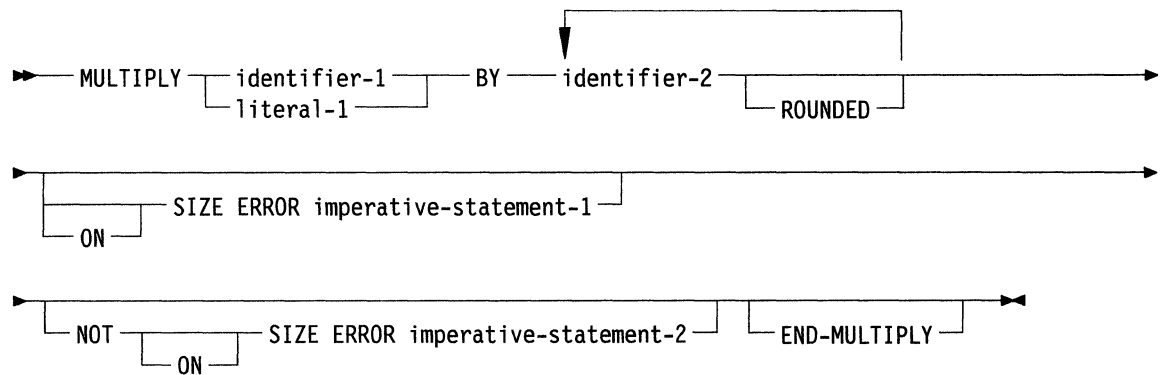
Function

The MULTIPLY statement causes numeric data items to be multiplied and sets the values of data items equal to the results.

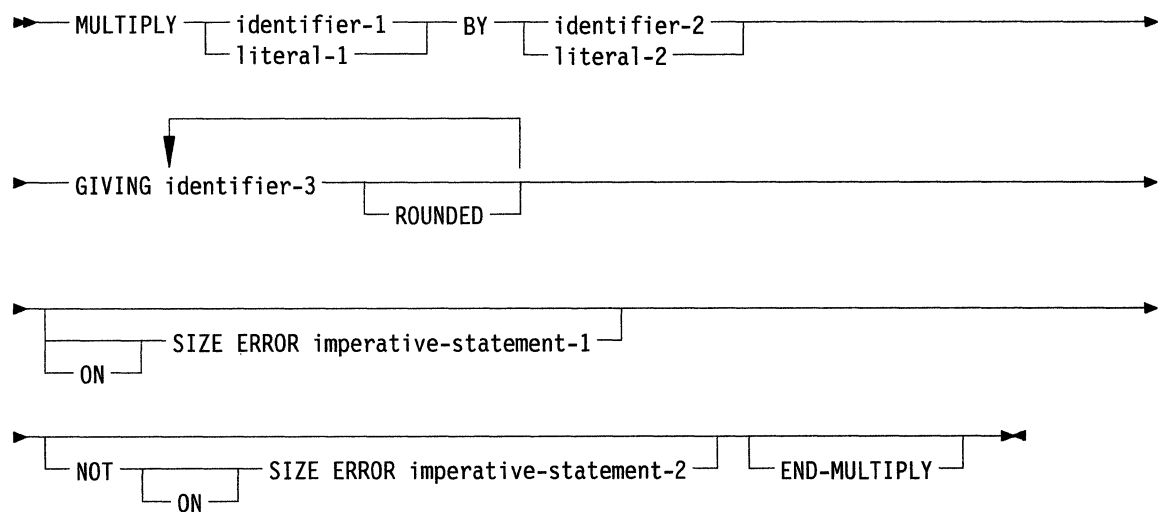
General Format

The following figures show the general format of the MULTIPLY statement:

Format 1



Format 2



Syntax Rules

The following syntax rules apply to the MULTIPLY statement:

1. Each identifier must refer to a numeric elementary item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric-edited item.
2. Each literal must be a numeric literal.
3. The composite of operands, which is that hypothetical data item resulting from the superimposition of all receiving data items aligned on their decimal points, must not contain more than 18 digits.

General Rules

The following general rules apply to the MULTIPLY statement:

1. Refer to “ROUNDED Phrase” on page 7-19, “ON SIZE ERROR Phrase and NOT ON SIZE ERROR Phrase” on page 7-19, “Arithmetic Statement Rules” on page 7-20, “Overlapping Operand Rules” on page 7-21, and “Multiple Results in Arithmetic Statement Rules” on page 7-21.
2. When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product (similarly for each successive occurrence of identifier-2 in the left-to-right order in which identifier-2 is specified).
3. When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2, and the result is stored in the data item referenced by each identifier-3.

ON Statement

Function

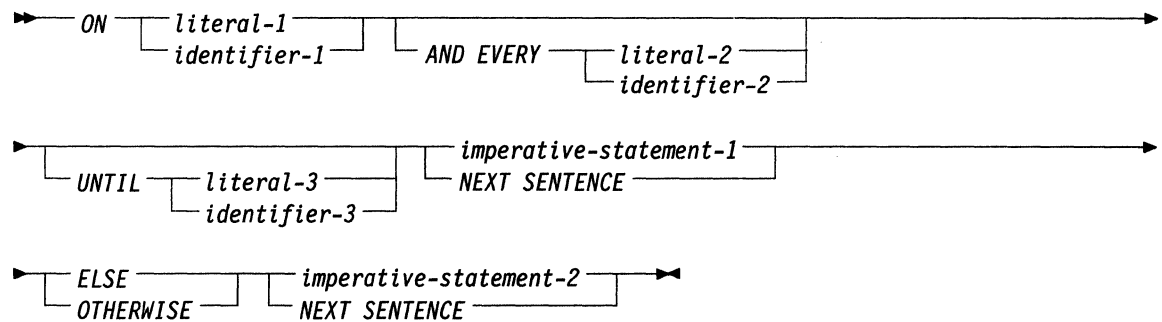
The ON statement allows selective execution of procedural statements on a periodic basis.

OSVS

General Format

The following figure shows the general format of the ON statement:

OSVS



Syntax Rules

The following syntax rules apply to the ON statement:

OSVS

- 1. identifier-1, identifier-2, and identifier-3 must describe unsigned integer numeric elementary items.*
- 2. literal-1, literal-2, and literal-3 must be unsigned numeric literals.*

General Rules

The following general rules apply to the ON statement:

OSVS

- 1. Prior to the first execution of each ON statement in the program, a counter, implicitly defined for that ON statement (the implicit-ON-counter), is initialized to be zero.*
- 2. identifier-1, identifier-2, and identifier-3 should, if specified, contain positive integer values at the time of execution of the ON statement. Varying these values between executions of the ON statement will affect subsequent executions of the ON statements.*

-
3. *The implicit ON counter cannot be affected in any way other than by transfer of program execution flow to that ON statement. (The ON counter of a CALLED program may only be reset by the CANCELing of that program; execution of the EXIT PROGRAM statement and subsequent CALL of the program without intervening CANCEL has no effect upon the implicit ON counter value.)*
 4. *The following value list is then evaluated:*
 - a. *The current value of identifier-1 or literal-1,*
 - b. *A sequence of values being the results of repeatedly adding the current value of identifier-2 or literal-2 to the current value of identifier-1 or literal-1 until the value of identifier-3 or literal-3 is reached.*

The implicit-ON-counter is then compared with each of this list of values. If an equality is found, then imperative-statement-1 is executed. If no equality is found, then imperative-statement-2 is executed.

Example

The following example shows the ON statement:

```
ON 1 AND EVERY 10 UNTIL 200
  DISPLAY "TRACE INFO:"
  EXHIBIT NAMED TRAN-COUNT
    TRAN-AMT.
```

PERFORM Statement

Function

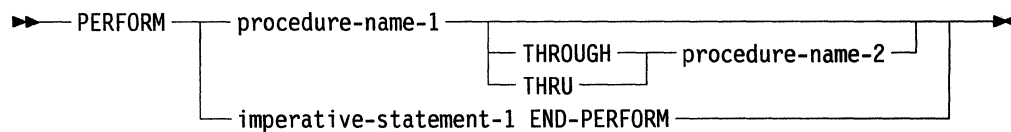
The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete.

The PERFORM statement is also used to control execution of one or more imperative statements that are within the scope of that PERFORM statement.

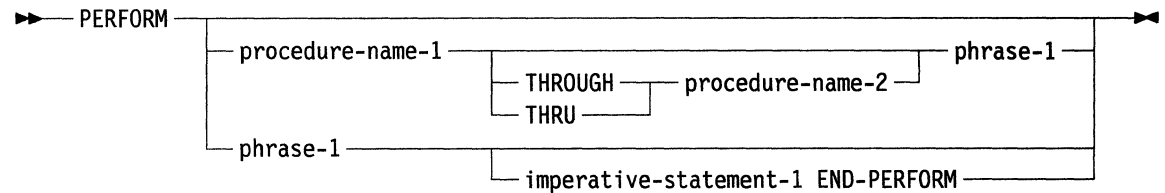
General Format

The following figures show the general format for the PERFORM statement:

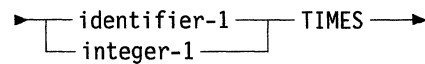
Format 1



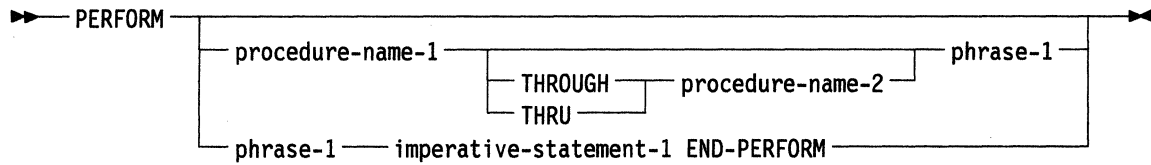
Format 2



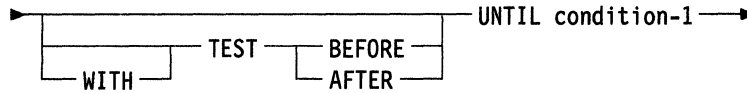
where phrase-1 is:



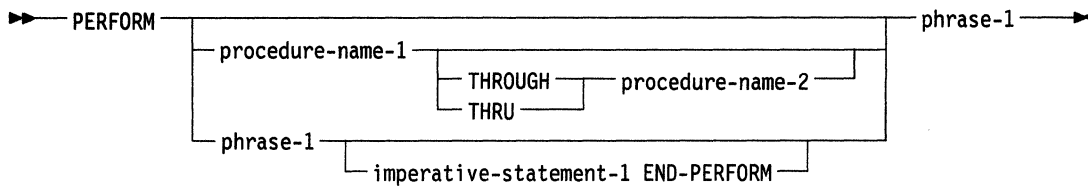
Format 3



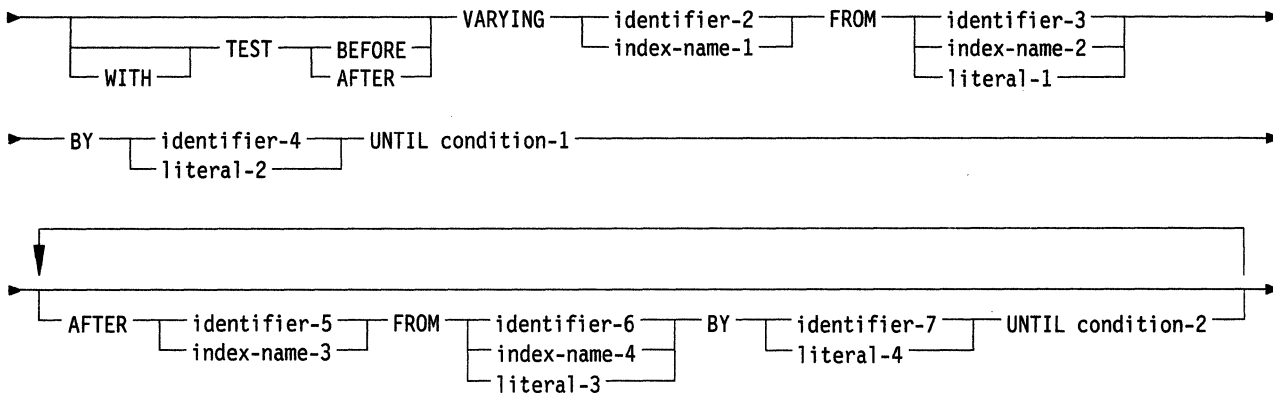
where phrase-1 is:



Format 4



where phrase-1 is:



Syntax Rules

The following syntax rules apply to the PERFORM statement:

1. If procedure-name-1 is omitted, imperative-statement-1 and the END-PERFORM phrase must be specified; if procedure-name-1 is specified, imperative-statement-1 and the END-PERFORM phrase must not be specified.
2. In Format 4, if procedure-name-1 is omitted, the AFTER phrase must not be specified.
3. If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.
4. Each identifier represents a numeric elementary item described in the Data Division. In Format 2, identifier-1 must be described as a numeric integer.
5. Each literal represents a numeric literal.

-
6. The words THROUGH and THRU are equivalent.
 7. If an index-name is specified in the VARYING or AFTER phrase, then:
 - a. The identifier in the associated FROM and BY phrases must reference an integer data item.
 - b. The literal in the associated FROM phrase must be a positive integer.
 - c. The literal in the associated BY phrase must be a nonzero integer.
 8. If an index name is specified in the FROM phrase, then:
 - a. The identifier in the associated VARYING or AFTER phrase must reference an integer data item.
 - b. The identifier in the associated BY phrase must reference an integer data item.
 - c. The literal in the associated BY phrase must be an integer.
 9. Literal in the BY phrase must not be zero.
 10. condition-1, condition-2, ..., may be any conditional expression. Refer to "Conditional Expressions" on page 7-9.
 11. Where procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the declaratives portion of the Procedure Division, both must be procedure names in the same declarative section.
 12. Six AFTER phrases are permitted in Format 4 of the PERFORM statement.

General Rules

The following general rules apply to the PERFORM statement:

1. The data items referenced by identifier-4 and identifier-7 must not have a zero value.
2. If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, the data item referenced by the identifier must have a positive value.
3. When procedure-name-1 is specified, the PERFORM statement is referred to as an out-of-line PERFORM statement; when procedure-name-1 is omitted, the PERFORM statement is referred to as an in-line PERFORM statement.
4. The statements contained within the range of procedure-name-1 (through procedure-name-2 if specified) for an out-of-line PERFORM statement, or contained within the PERFORM statement itself for an in-line PERFORM statement are referred to as the specified set of statements.
5. The END-PERFORM phrase delimits the scope of the in-line PERFORM statement. Refer to "Explicit and Implicit Scope Terminators" on page 2-36.
6. An in-line PERFORM statement functions according to the following general rules for an otherwise identical out-of-line PERFORM statement, with the exception that the statements contained within the in-line PERFORM statement 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 general rules that apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.
7. When the PERFORM statement is executed, control is transferred to the first statement of the specified set of statements (except as indicated in 10b on page 7-76, 10c on page 7-76, and 10d on page 7-77). This transfer of control occurs only once for each execution of a PERFORM statement. For those cases where a transfer of control to the specified set of statements does take place, an implicit transfer of control to the end of the PERFORM statement is established as follows:

-
- a. If procedure-name-1 is a paragraph name and procedure-name-2 is not specified, the return is after the last statement of procedure-name-1.
 - b. If procedure-name-1 is a section name and procedure-name-2 is not specified, the return is after the last statement of the last paragraph in procedure-name-1.
 - c. If procedure-name-2 is specified and it is a paragraph name, the return is after the last statement of the paragraph.
 - d. If procedure-name-2 is specified and it is a section name, the return is after the last statement of the last paragraph in the section.
 - e. If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained with it has been executed.
8. There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement, to which all of these paths must lead.
 9. If control passes to the specified set of statements by means other than a PERFORM statement, control will pass through the last statement of the set to the next executable statement as if no PERFORM statement referenced the set.
 10. The PERFORM statements operate as follows:
 - a. Format 1 is the basic PERFORM statement. The specified set of statements referenced by this type of PERFORM statement is executed once and then control passes to the end of the PERFORM statement.
 - b. Format 2 is the PERFORM ... TIMES. The specified set of statements is performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. If at the time of the execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is negative, control passes to the end of the PERFORM statement. Following the execution of the specified set of statements the specified number of times, control is transferred to the end of the PERFORM statement.

During execution of the PERFORM statement, reference to identifier-1 cannot alter the number of times the specified set of statements is to be executed from that which was indicated by the initial value of the data item referenced by identifier-1.
 - c. Format 3 is the PERFORM ... UNTIL. The specified set of statements is performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the end of the PERFORM statement. If the condition is true when the PERFORM statement is entered, the TEST BEFORE phrase is specified or implied, no transfer to procedure-name-1 takes place, and control is passed to the end of the PERFORM statement.

If the TEST AFTER phrase is specified, the PERFORM statement functions as if the TEST BEFORE phrase were specified, except that the condition is tested after the specified set of statements has been executed. Any subscripting or reference modification associated with the operands specified in condition-1 is evaluated each time the condition is tested.

-
- d. Format 4 is the PERFORM ... VARYING. This variation of the PERFORM statement is used to augment the values referenced by one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING, AFTER, and FROM (current value) phrases also refers to index-names. If index-name-1 or index-name-3 is specified, the value of the associated index at the beginning of the PERFORM statement must be set to an occurrence number of an element in the table. If index-name-2 or index-name-4 is specified, the value of the data item referenced by identifier-2 or identifier-5 at the beginning of the PERFORM statement must be equal to an occurrence number of an element in a table associated with index-name-2 or index-name-4.

Subsequent augmentation, as described below, of index-name-1 or index-name-3 must not result in the associated index being set to a value outside the range of the table associated with index-name-1 or index-name-3. At the completion of the PERFORM statement, the index associated with index-name-1 may contain a value that is outside the range of the associated table by one increment or decrement value. 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 augmenting operation. Any subscripting or reference modification associated with the operands specified in condition-1 or condition-2 is evaluated each time the condition is tested.

Representations of the actions of several types of Format 4 PERFORM statements are given on the following pages.

- 1) If the TEST BEFORE phrase is specified or implied:

When the data item referenced by one identifier is varied, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3 at the point of initial execution of the PERFORM statement. Then, if the condition of the UNTIL phrase is false, the specified set of statements is executed once. The value of the data item referenced by identifier-2 is augmented by the specified increment or decrement value (literal-2 or the value of the data item referenced by identifier-4) and condition-1 is evaluated again. The cycle continues until this condition is true, at which point control is transferred to the end of the PERFORM statement. If condition-1 is true at the beginning of execution of the PERFORM statement, control is transferred to the end of the PERFORM statement as shown in Figure 7-3.

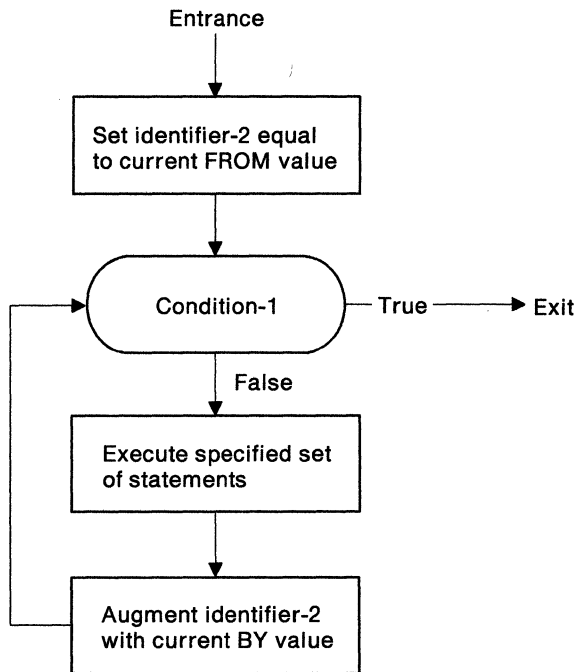


Figure 7-3. The VARYING Option of a PERFORM Statement with the TEST BEFORE Phrase Having One Condition

When the data items referenced by two identifiers are varied, the content of the data item referenced by identifier-2 is set to literal-1, or the current value of the data item referenced by identifier-3, and then the content of the data item referenced by identifier-5 is set to literal-3, or the current value of the data item referenced by identifier-6. After the contents of the data items referenced by the identifiers have been set, condition-1 is evaluated; if true, control is transferred to the end of the PERFORM statement; if false, condition-2 is evaluated. If condition-2 is false, the specified set of statements is executed once, and the content of the data item referenced by identifier-5 is then augmented by literal-4 or the content of the data item referenced by identifier-7, and condition-2 is evaluated again.

This cycle of evaluation and augmentation continues until the condition is true. When condition-2 is true, the content of the data item referenced by identifier-2 is augmented by literal-2 or the content of the data item referenced by identifier-4. The content of the data item referenced by identifier-5 is set to literal-3, or the current value of the data item referenced by identifier-6, and condition-1 is reevaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycle continues until condition-1 is true.

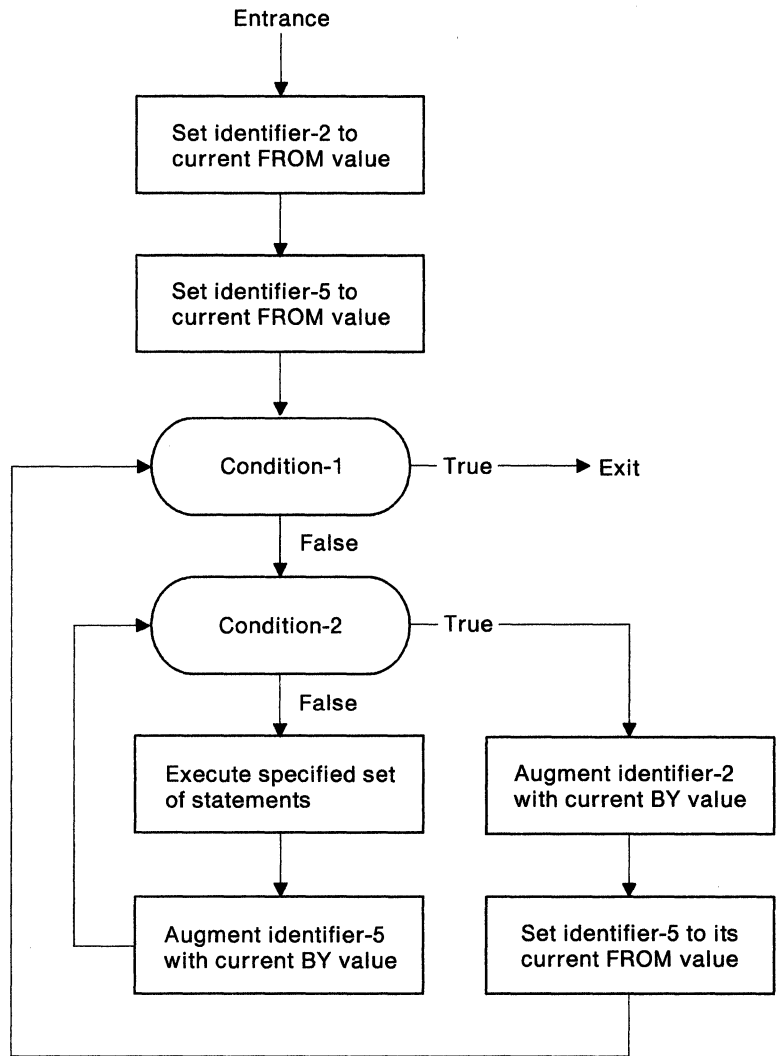


Figure 7-4. The VARYING Option of a PERFORM Statement with the TEST BEFORE Phrase Having Two Conditions

At the termination of the PERFORM statement, the data item referenced by identifier-5 contains literal-3 or the current value of the data item referenced by identifier-6. The data item referenced by identifier-2 contains a value that exceeds the last used setting by one increment or decrement value, unless condition-1 was true when the PERFORM statement was entered, in which case the data item referenced by identifier-2 contains literal-1 or the current value of the data item referenced by identifier-3.

2) If the TEST AFTER phrase is specified:

When the data item referenced by one identifier is varied, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3 at the point of execution of the PERFORM statement; then the specified set of statements is executed once and condition-1 of the UNTIL phrase is tested. If the condition is false, the value of the data item referenced by identifier-2 is augmented by the specified increment or decrement value (literal-2 or the value of the data item referenced by identifier-4) and the specified set of statements is executed again. The cycle continues until condition-1 is tested and found to be true, at which point control is transferred to the end of the PERFORM statement.

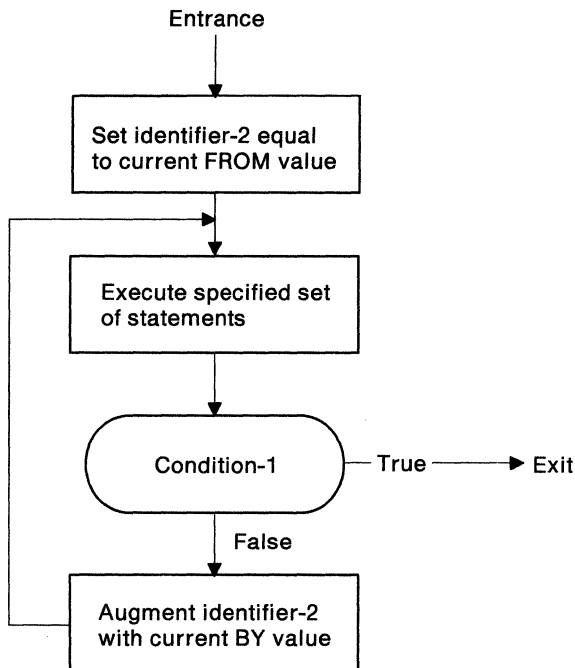


Figure 7-5. The VARYING Option of a PERFORM Statement with the TEST AFTER Phrase Having One Condition

When the data items referenced by two identifiers are varied, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3. The current value of the data item referenced by identifier-5 is then set to literal-3 or the current value of the data item referenced by identifier-6 and the specified set of statements is then executed. condition-2 is then evaluated. If false, the content of the data item referenced by identifier-5 is augmented by literal-4 or the content of the data item referenced by identifier-7, and the specified set of statements is again executed.

The cycle continues until condition-2 is again evaluated and found to be true, at which time condition-1 is evaluated. If false, the content of the data item referenced by identifier-2 is augmented by literal-2 or the content of data item referenced by identifier-4. The content of the data item referenced by identifier-5 is set to literal-3, or the current value of the data item referenced by identifier-6, and the specified set of statements is again executed. This cycle continues until condition-1 is again evaluated and found to be true, at which time control is transferred to the end of the PERFORM statement.

After completion of the PERFORM statement, each data item varied by an AFTER or VARYING phrase contains the same value it contained at the end of the most recent execution of the specified set of statements.

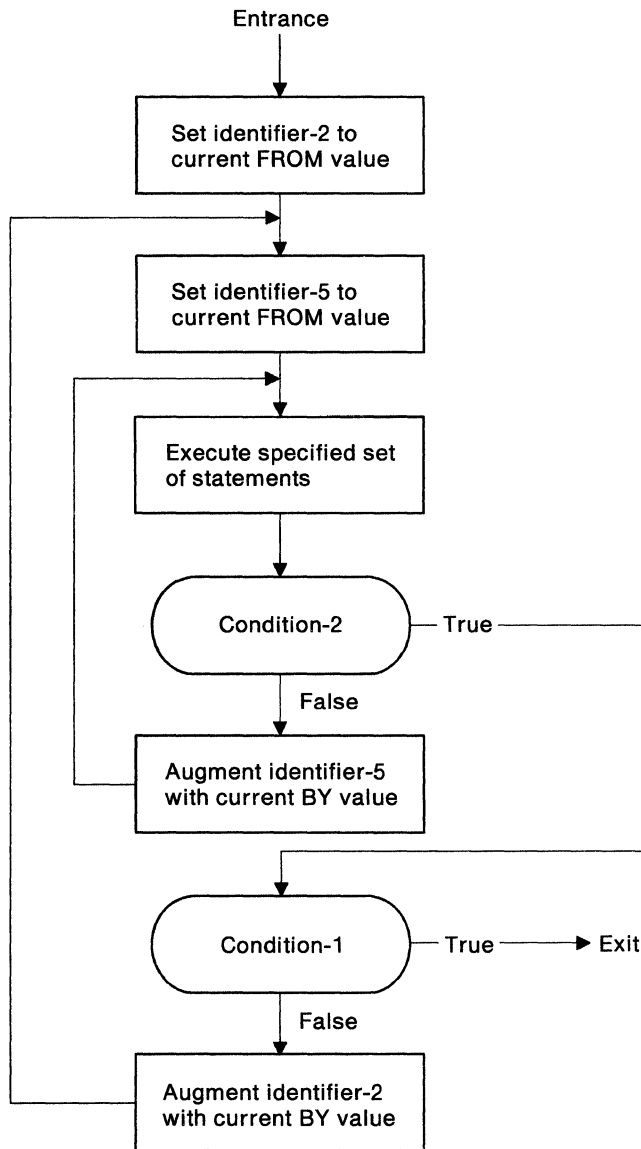


Figure 7-6. The VARYING Option of a PERFORM Statement with a TEST AFTER Phrase Having Two Conditions

During the execution of the specified set of statements associated with the PERFORM statement, any change to the VARYING variable (the data item referenced by identifier-2 and index-name-1), the BY variable (the data item referenced by identifier-4), the AFTER variable (the data item referenced by identifier-5 and index-name-3), or the FROM variable (the data item referenced by identifier-3 and index-name-2) will be taken into consideration and will affect the operation of the PERFORM statement.

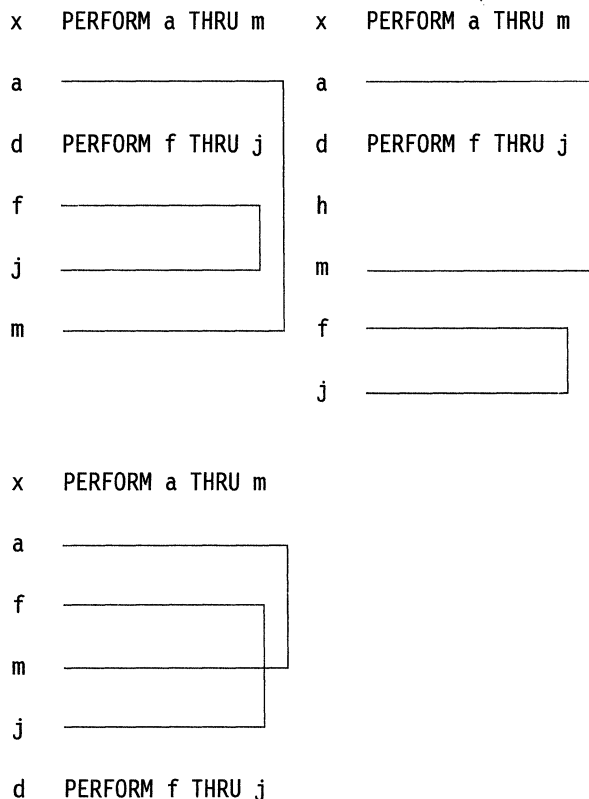
When the data items referenced by two identifiers are varied, the data item referenced by identifier-5 goes through a complete cycle (FROM, BY, UNTIL) each time the content of the data item referenced by identifier-2 is varied. When the contents of three or more data items referenced by identifiers are varied, the mechanism is the same as for two identifiers except that the data item being varied by each AFTER phrase goes through a complete cycle each time the data item being varied by the preceding AFTER phrase is augmented.

11. The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the explicit transfer of control to the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the PERFORM statement, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source program.
12. Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement within the range of a PERFORM statement are not considered to be part of that range.
13. If the range of a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM should itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, should not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements should not have a common exit.

MF

These restrictions are not enforced. PERFORM statements can be freely nested, and recursion (a PERFORM statement performing a procedure containing it) is allowed. Only the exit-point of the innermost PERFORM statement currently being executed is recognized. These rules can be changed by use of a COBOL system directive. Refer to the User's Guide.

The following illustrations show examples of legal PERFORM constructs:



14. A PERFORM statement that appears in a section which is not in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, one of the following:

- a. Sections and/or paragraphs wholly contained in one or more nonindependent segments.
- b. Sections and/or paragraphs wholly contained in a single independent segment.

These restrictions do not apply.

OSVS VSC2

15. A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

- a. Sections and/or paragraphs wholly contained in one or more nonindependent segments.
- b. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

These restrictions do not apply.

OSVS VSC2

SET Statement

Function

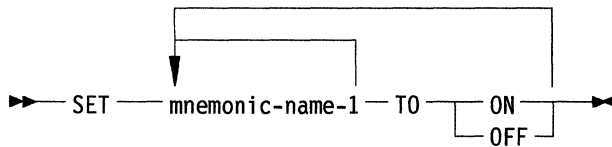
The SET statement performs the following functions:

1. The SET statement is used to alter the status of external switches.
2. The SET statement is used to alter the value of conditional variables.
3. *The SET statement is used to assign the address of a data item to a pointer variable.* VSC2
4. The SET statement also establishes reference points for table-handling operations by setting indices associated with table elements. Refer to "SET Statement" on page 12-19.

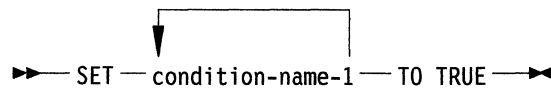
General Format

The following figures show the general format of the SET statement:

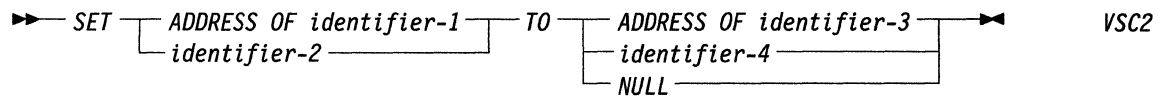
Format 1



Format 2



Format 3



Syntax Rules

The following syntax rules apply to the SET statement:

1. mnemonic-name-1 must be associated with an external switch, the status of which can be altered.
2. condition-name-1 must be associated with a conditional variable.

-
3. *identifier-1 must refer to the 01 to 77 level item in the LINKAGE SECTION. identifier-3 must refer to the 01 or 77 level item in the LINKAGE or WORKING-STORAGE SECTION. identifier-2 and identifier-4 must refer to an item with USAGE IS POINTER.* VSC2 MF

General Rules

The following general rules apply to the SET statement:

Format 1

1. The status of each external switch associated with the specified mnemonic-name-1 is modified such that the truth value resulting from evaluation of a condition-name associated with that switch will reflect an on status if the ON phrase is specified, or an off status if the OFF phrase is specified. Refer to "Switch-Status Condition" on page 7-15.

Format 2

2. The literal in the VALUE clause associated with condition-name-1 is placed in the conditional variable according to the rules of the VALUE clause. Refer to "VALUE Clause" on page 6-41. If more than one literal is specified in the VALUE clause, the conditional variable is set to the value of the first literal that appears in the VALUE clause.
3. If multiple condition-names are specified, the results are the same as if a separate SET statement had been written for each condition-name-1 in the same order as specified in the SET statement.

Format 3

4. *If identifier-2 is specified, the address given by the TO phrase is moved into identifier-2. If identifier-1 is specified, identifier-1 is relinked so that any subsequent reference to it references the item whose address is given by the TO phrase. The address given by the TO phrase is the address contained in identifier-4, or the address of identifier-3, or, if NULL is specified, a null address guaranteed to point to no data item. Refer to "USAGE Clause" on page 6-39.* VSC2

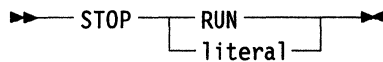
STOP Statement

Function

The STOP statement causes a permanent or temporary suspension of the execution of the object program.

General Format

The following figure shows the general format for the STOP statement:



Syntax Rules

The following syntax rules apply to the STOP statement:

1. The literal may be numeric, nonnumeric, or any figurative constant except ALL.
2. If the literal is numeric, then it must be an unsigned integer.
3. If a STOP RUN statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules

The following general rules apply to the STOP statement:

1. If the RUN phrase is used, then the operating system ending procedure is instituted.
2. If STOP literal is specified, the literal is communicated to the operator. Continuation of the object program begins with the execution of the next executable statement in sequence. Pressing the RETURN key or equivalent continues execution.

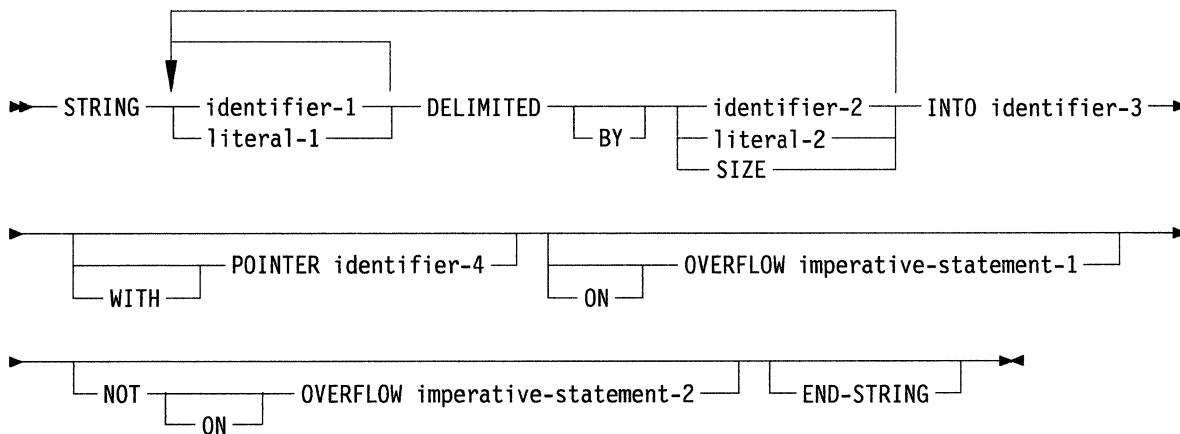
STRING Statement

Function

The **STRING** statement provides juxtaposition of the partial or complete contents of two or more data items into a single data item.

General Format

The following figure shows the general format of the **STRING** statement:



Syntax Rules

The following syntax rules apply to the **STRING** statement:

1. Each literal may be any figurative constant without the optional word **ALL**.
2. All literals must be described as nonnumeric literals, and all identifiers, except identifier-4, must be described implicitly or explicitly as usage is **DISPLAY**.
3. identifier-3 must represent an elementary alphanumeric data item without editing symbols or the **JUSTIFIED** clause.
4. identifier-4 must represent an elementary numeric integer data item of sufficient size to contain a value equal to the size plus one of the area referred to by identifier-3. The **P** may not be used in the **PICTURE** character-string of identifier-4.
5. Where identifier-1 or identifier-2 is an elementary numeric data item, it must be described as an integer without the symbol **P** in its **PICTURE** character-string.
6. identifier-3 must not be reference modified.

DBCS Support

7. identifier-1 through identifier-3 may be DBCS (USAGE DISPLAY-1) items. If one of these identifiers is a DBCS item, then all of them must be DBCS items.
8. literal-1 through literal-2 may be DBCS literals. If any one is a DBCS literal, then all of them must be DBCS literals.
9. SPACE and SPACES are the only figurative constants allowed for DBCS items.
10. When identifier-3 (the receiving field) is a DBCS data item, identifier-4 indicates the relative DBCS character position in the receiving field.

End of DBCS Support

General Rules

The following general rules apply to the STRING statement:

1. identifier-1, or literal-1, represents the sending item. identifier-3 represents the receiving item.
2. literal-2, or identifier-2, indicate the character(s) delimiting the move. If the SIZE phrase is used, the complete data item defined by identifier-1, or literal-1, is moved. When a figurative constant is used as the delimiter, it stands for a single character nonnumeric literal.
3. When a figurative constant is specified as literal-1, or literal-2, it refers to an implicit one-character data item whose usage is DISPLAY.
4. When the STRING statement is executed, the transfer of data is governed by the following rules:
 - a. Those characters from literal-1, or from the contents of the data item referred to by identifier-1, are transferred to the contents of identifier-3 in accordance with the rules for alphanumeric to alphanumeric moves, except that no space filling will be provided. Refer to "MOVE Statement" on page 7-65.
 - b. If the DELIMITED phrase is specified without the SIZE phrase, the contents of the data item referred to by identifier-1, or the value of literal-1, are transferred to the receiving data item in the sequence specified in the STRING statement. Transfer begins with the leftmost character and continues from left to right until the end of the data item is reached or until the character(s) specified by literal-2 or by the contents of identifier-2 are encountered. The character(s) specified by literal-2, or by the data item referred to by identifier-2, are not transferred.
 - c. If the DELIMITED phrase is specified with the SIZE phrase, the entire contents of literal-1, or the contents of the data item referred to by identifier-1, are transferred, in the sequence specified in the STRING statement, to the data item referred to by identifier-3 until all data has been transferred or the end of the data item referred to by identifier-3 has been reached.
5. If the POINTER phrase is specified, identifier-4 is explicitly available to the programmer, who is then responsible for setting its initial value. The initial value must not be less than one.
6. If the POINTER phrase is not specified, the following general rules apply as if the user had specified identifier-4 with an initial value of one.
7. When characters are transferred to the data item referred to by identifier-3, it is as if the characters are moved one at a time from the source into the character position of the data item referred to by identifier-3 (designated by the value associated with identifier-4). By this means identifier-4 is increased by one prior to the move of the next character. The value associated with identifier-4 is changed during execution of the STRING statement only in the manner specified above.

-
8. Only the portion of the data item referred to by identifier-3 during the execution of the STRING statement is changed at the end of the execution of the STRING statement. All other portions of the data item referred to by identifier-3 will contain data that was present before this execution of the STRING statement.
 9. If, prior to each move of a character to the data item referred to by identifier-3, the value associated with the data item referred to by identifier-4 is either less than one or exceeds the number of character positions in the data item referred to by identifier-3, no further data is transferred to the data item referred to by identifier-3. The NOT ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the STRING statement. If the ON OVERFLOW phrase is specified, control is transferred 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 in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the STRING statement.

10. If, at the time of execution of a STRING statement with the NOT ON OVERFLOW phrase, the conditions described in this step are not encountered, after completion of the transfer of data according to the other general rules, the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the STRING statement or, if the NOT ON OVERFLOW phrase is specified, to imperative-statement-2.

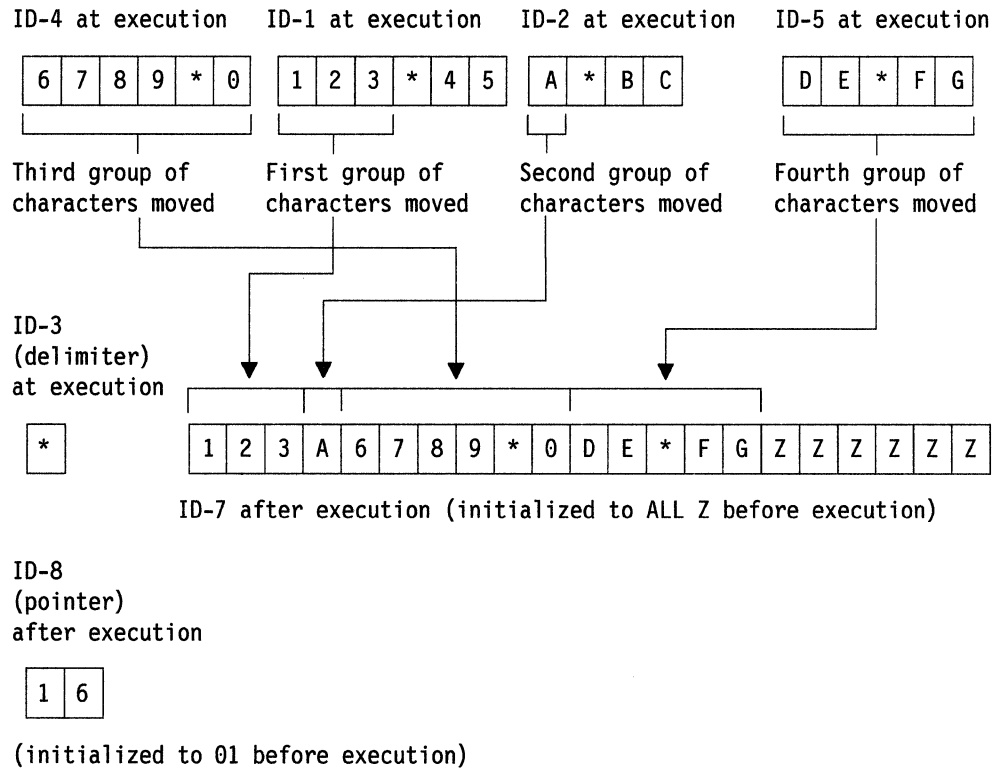
If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the STRING statement.

11. The END-STRING phrase delimits the scope of the STRING statement. Refer to "Explicit and Implicit Scope Terminators" on page 2-36.

Example

The following example shows the execution of a STRING statement and its result:

```
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.
```



SUBTRACT Statement

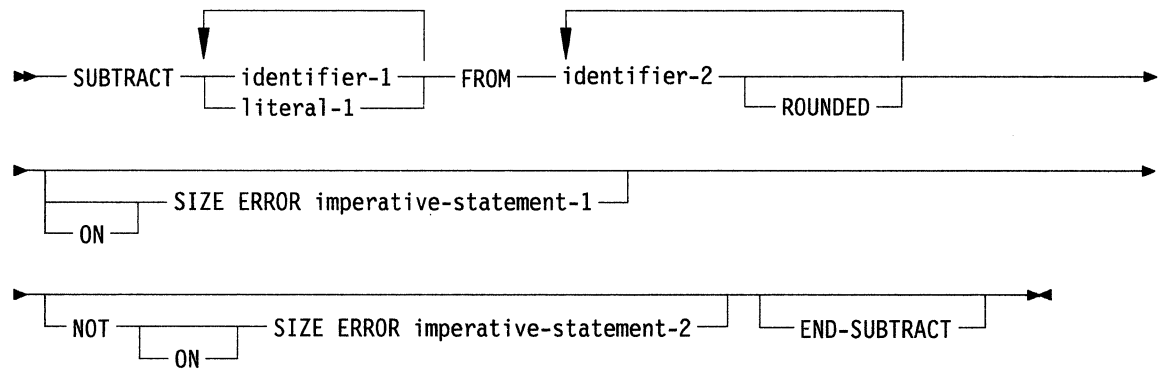
Function

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more items equal to the results.

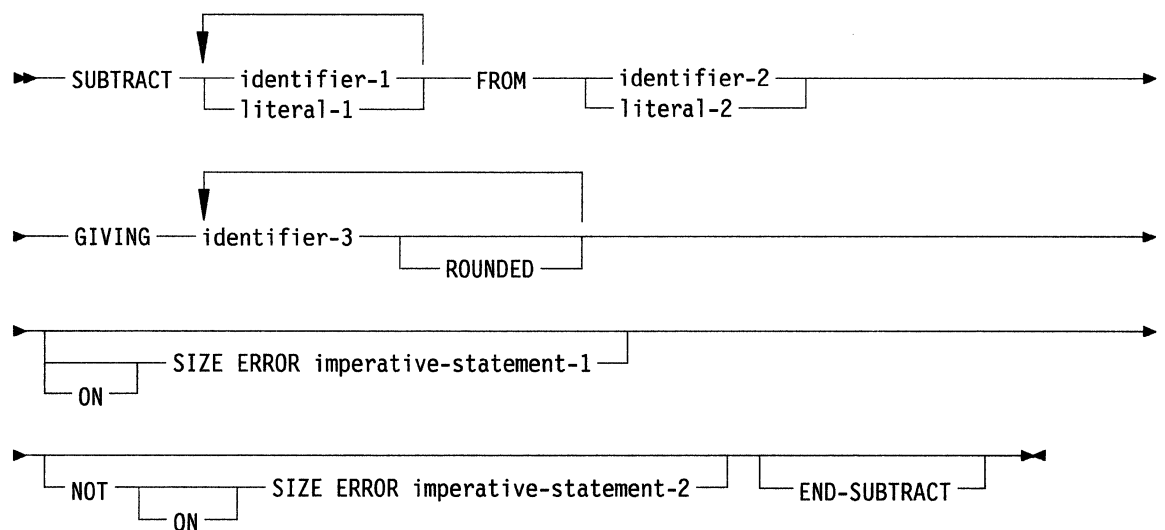
General Format

The following figures show the general format of the SUBTRACT statement:

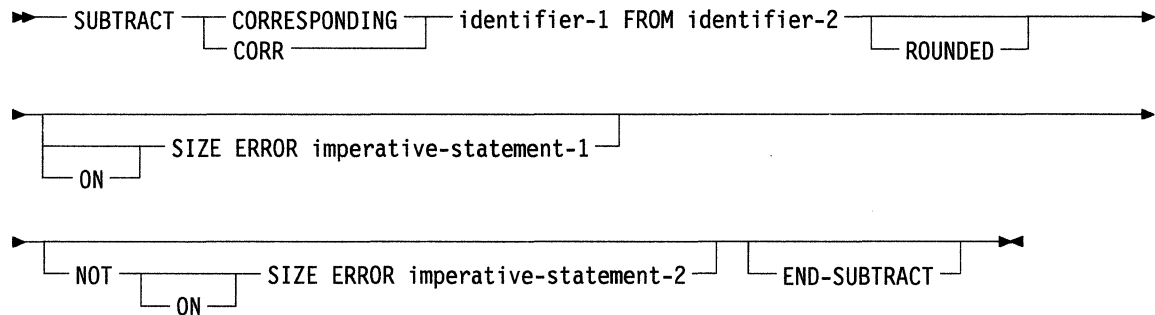
Format 1



Format 2



Format 3



Syntax Rules

The following syntax rules apply to the SUBTRACT statement:

1. Each identifier must refer to a numeric elementary item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric-edited item, and in Format 3 each identifier must refer to a group item.
2. Each literal must be a numeric literal.
3. The composite of operands must not contain more than 18 digits. Refer to “Arithmetic Statement Rules” on page 7-20.
 - a. In Format 1, the composite of operands is determined by using all of the operands in a given statement.
 - b. 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.
 - c. In Format 3, the composite of operands is determined separately for each pair of corresponding data items.
4. CORR is an abbreviation for CORRESPONDING.

General Rules

The following general rules apply to the SUBTRACT statement:

1. Refer to “ROUNDED Phrase” on page 7-19, “ON SIZE ERROR Phrase and NOT ON SIZE ERROR Phrase” on page 7-19, “Arithmetic Statement Rules” on page 7-20, “Overlapping Operand Rules” on page 7-21, and “Multiple Results in Arithmetic Statement Rules” on page 7-21.
2. In Format 1, all literals or identifiers preceding the word FROM are added together and this total is subtracted from the current value of identifier-2 storing the result immediately into identifier-2, and repeating this process respectively for each operand following the word FROM.
3. In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-2 or identifier-2, and the result of the subtraction is stored as the new value of each data item referred to by identifier-3.

-
4. If Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.
 5. AIX VS COBOL system ensures enough places are carried so as not to lose significant digits during execution.

TRANSFORM Statement

Function

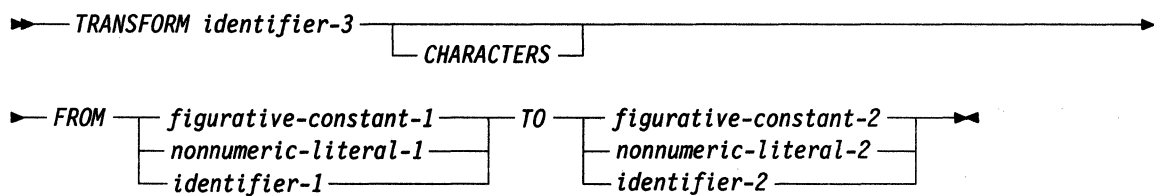
The *TRANSFORM* statement is used to alter characters according to a transformation rule.

OSVS

General Format

The following figure shows the general format of the *TRANSFORM* statement:

OSVS



Syntax Rules

The following syntax rules apply to the *TRANSFORM* statement:

OSVS

1. *identifier-3* may be any elementary item except a numeric item with *USAGE* other than *DISPLAY*, or a group item.
2. *identifier-1* and *identifier-2* should be elementary alphabetic or alphanumeric items.
3. *nonnumeric-literal-2* or *identifier-2* must be either one character long or the same length as *nonnumeric-literal-1* or *identifier-1*.

General Rules

The following general rules apply to the TRANSFORM statement:

OSVS

- 1. The use of figurative-constant-1 and/or figurative-constant-2 is equivalent to the use of a single-character nonnumeric-literal with the same value.*
- 2. If either identifier-1 or identifier-2 refers to the same computer storage area as identifier-3, the result of the TRANSFORM statement will be undefined. Refer to "REDEFINES Clause" on page 6-29.*
- 3. If characters are repeated in nonnumeric-literal-1 or identifier-1, then the result of the TRANSFORM operation is undefined.*
- 4. Execution of the TRANSFORM statement scans identifier-3 for occurrences of individual characters from identifier-1 or nonnumeric-literal-1. When a match is found, the corresponding character (or the single character of a one-character field) from identifier-2 or nonnumeric-literal-2 is substituted into that character position in identifier-3. The correspondence between identifier-1 or nonnumeric-literal-1 and identifier-2 or nonnumeric-literal-2 is by occurrence number of the character within the data item (starting from the left).*

Example

The following example shows the TRANSFORM statement:

```
TRANSFORM LETTER-ITEM CHARACTERS  
FROM "ABCDEF" TO "123456".
```

LETTER-ITEM

before execution	F	X	V	E	D	C	B	A	F	B
after execution	6	X	V	5	4	3	2	1	6	2

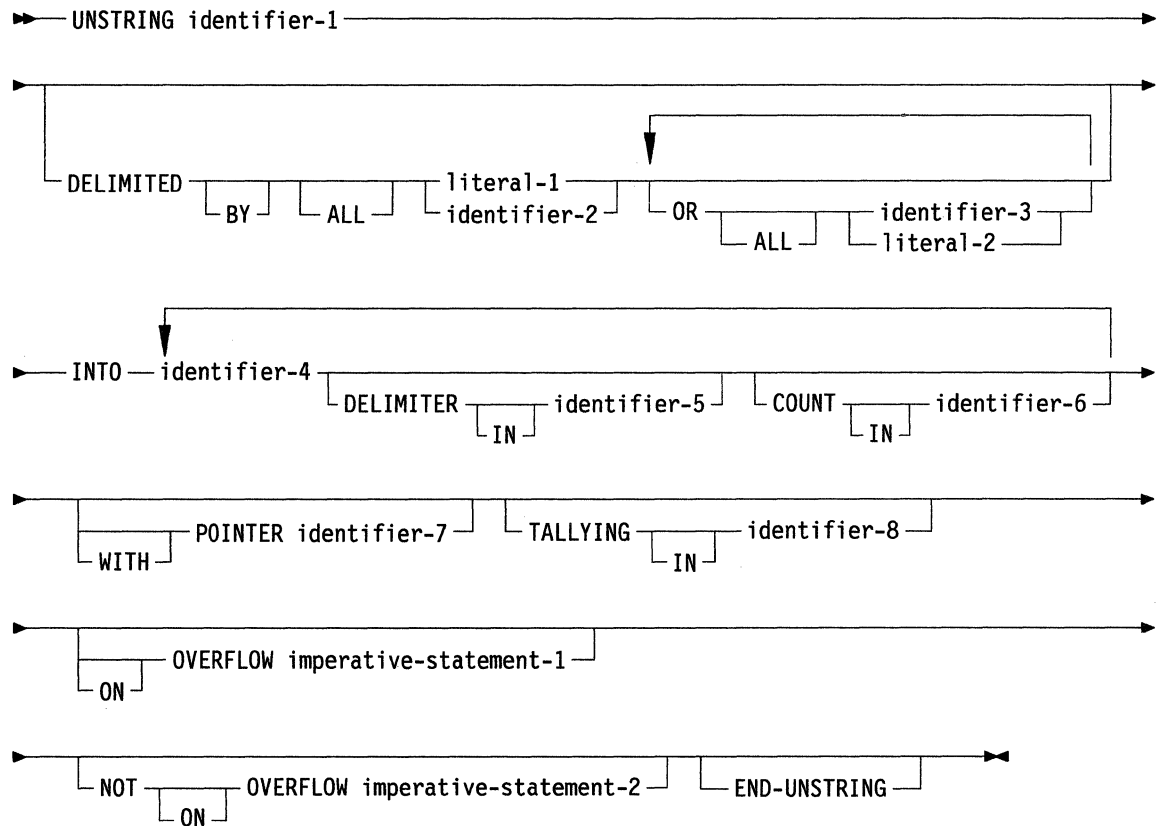
UNSTRING Statement

Function

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

General Format

The following figure shows the general format for the UNSTRING statement:



Syntax Rules

The following syntax rules apply to the UNSTRING statement:

1. Each literal must be a nonnumeric literal. In addition, each literal may be any figurative constant without the optional word ALL.
2. identifier-1, identifier-2, identifier-3, and identifier-5 must be described, implicitly or explicitly, as an alphanumeric data item.

-
3. identifier-4 may be described as either alphabetic (except that the symbol B may not be used in the PICTURE character-string), alphanumeric, or numeric (except that the symbol P may not be used in the PICTURE character-string), and must be described as USAGE IS DISPLAY.
 4. identifier-6, identifier-7, and identifier-8 must be described as elementary numeric integer data items, except that the symbol P may not be used in the PICTURE character-string.
 5. No identifier may name a level 88 entry.
 6. The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

DBCS Support

7. identifier-1 can be a DBCS data item.
8. identifier-2 and identifier-3 can be DBCS data items. If any are DBCS items, then all must be DBCS items. Figurative constants SPACE and SPACES are allowed for DBCS items.
9. literal-1 and literal-2 can be DBCS literals. If any are DBCS literals, then all must be DBCS literals. Figurative constants SPACE and SPACES are allowed for DBCS literals.
10. identifier-4 can be a DBCS data item.
11. identifier-5 can be a DBCS data item.

End of DBCS Support

General Rules

The following general rules apply to the UNSTRING statement:

1. All references to identifier-2, literal-1, apply equally to identifier-3, literal-2, respectively, and to all repetitions of these items.
2. identifier-1 represents the sending area.
3. identifier-4 represents the data receiving area. identifier-5 represents the receiving area for delimiters.
4. literal-1 or the data item referred to by identifier-2 specifies a delimiter.
5. The data item referred to by identifier-6 represents the count of the number of characters within the data item referred to by identifier-1 isolated by the delimiters for the move to the data item referred to by identifier-4. This value does not include a count of the delimiter character(s).
6. The data item referred to by identifier-7 contains a value that indicates a relative character position within the area defined by identifier-1.
7. The data item referred to by identifier-8 is a counter that records the number of data items acted upon during the execution of an UNSTRING statement.
8. When a figurative constant is used as the delimiter, it stands for a single character nonnumeric literal.

When the ALL phrase is specified, one occurrence, or two or more contiguous occurrences of literal-1 (figurative constant or not), or the contents of the data item referred to by identifier-2 are treated as if it were only one occurrence. This occurrence is moved to the receiving data item according to the rules in 13d on page 7-98.

9. When any examination encounters two contiguous delimiters, the current receiving area is either space or zero-filled according to the description of the receiving area.

-
10. literal-1 or the contents of the data item referred to by identifier-2 can contain any character in the character set.
 11. Each literal-1 or the data item referred to by identifier-2 represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item and in the order given, to be recognized as a delimiter.
 12. When two or more delimiters are specified in the DELIMITED BY phrase, an OR condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field can be considered a part of more than one delimiter.

Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

13. When the UNSTRING statement is initiated, the current receiving area is the data item referred to by identifier-4. Data is transferred from the data item referred to by identifier-1 to the data item referred to by identifier-4 according to the following rules:
 - a. If the POINTER phrase is specified, the string of characters referred to by identifier-1 is examined beginning with the relative character position indicated by the contents of the data item referred to by identifier-7. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.
 - b. If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by the value of literal-1 or the data item referred to by identifier-2 is encountered. Refer to rule 11 in this list. If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

If the end of the data item referred to by identifier-1 is encountered before the delimiting condition is met, the examination terminates with the last character examined.
 - c. The characters thus examined (excluding the delimiting characters, if any) are treated as an elementary alphanumeric data item and are moved into the current receiving area according to the rules for the MOVE statement. Refer to "MOVE Statement" on page 7-65.
 - d. If the DELIMITER IN phrase is specified, the delimiting characters are treated as an elementary alphanumeric data item and are moved into the data item referred to by identifier-5 according to the rules of the MOVE statement. Refer to "MOVE Statement" on page 7-65. If the delimiting condition is the end of the data item referred to by identifier-1, the data item referred to by identifier-5 is space-filled.
 - e. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter characters if any) is moved into the area referred to by identifier-6 according to the rules for an elementary move.
 - f. If the DELIMITED BY phrase is specified, the string of characters is further examined beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified, the string of characters is further examined beginning with the character to the right of the last character transferred.
 - g. After data is transferred to the data item referred to by identifier-4, the current receiving area is the data item referred to by the next recurrence of identifier-4. The steps described in rules 13b through 13f are repeated until either all the characters are exhausted in the data item referred to by identifier-1, or until there are no more receiving areas.
14. The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

-
15. The contents of the data item referred to by identifier-7 will be incremented by one for each character examined in the data item referred to by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is complete, the contents of the data item referred to by identifier-7 will contain a value equal to the initial value plus the number of characters examined in the data item referred to by identifier-1.
 16. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the contents of the data item referred to by identifier-8 contains a value equal to its initial value plus the number of data receiving items acted upon.
 17. Either of the following situations causes an overflow condition:
 - a. An UNSTRING is initiated, and the value in the data item referred to by identifier-7 is less than one or greater than the size of the data item referred to by identifier-1.
 - b. If, during execution of an UNSTRING statement, all data receiving areas have been acted upon, and the data item referred to by identifier-1 contains characters that have not been examined.
 18. When an overflow condition exists, the UNSTRING operation is terminated, the NOT ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the UNSTRING 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 in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the UNSTRING statement.
 19. The END-UNSTRING phrase delimits the scope of the UNSTRING statement. Refer to "Explicit and Implicit Scope Terminators" on page 2-36.
 20. If, at the time of execution of an UNSTRING statement, the conditions described in 17 are not encountered (after completion of the transfer of data according to the other general rules), the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the UNSTRING statement, or, if the NOT ON OVERFLOW phrase is specified, to imperative-statement-2.

If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the UNSTRING statement.
 21. The evaluation of subscripting and indexing for the identifiers is as follows:
 - a. Any subscripting or indexing associated with identifier-1, identifier-7, or identifier-8 is evaluated only once, immediately before any data is transferred as the result of the execution of the UNSTRING statement.
 - b. Any subscripting or indexing associated with identifier-2, identifier-3, identifier-4, identifier-5, or identifier-6 is evaluated immediately before the transfer of data into the respective data item.
 22. Any subscripting associated with the DELIMITED BY identifier, the INTO identifier, the DELIMITER IN identifier, or the COUNT IN identifier is evaluated once, immediately before the examination of the sending fields for the delimiter.

DBCS Support

23. 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.
24. When identifier-1 (the sending field) is a DBCS data item, identifier-7 indicates the relative DBCS character position in the sending field.

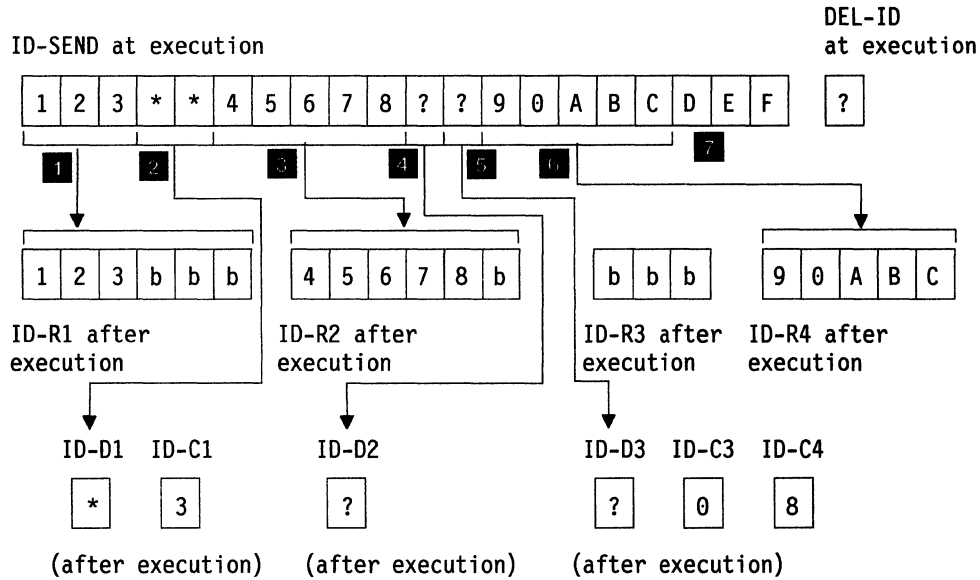
End of DBCS Support

Example

The following example shows the UNSTRING statement and the execution results:

```
UNSTRING ID-SEND DELIMITED BY DEL-ID OR ALL "*"
  INTO ID-R1 DELIMITER IN ID-D1 COUNT IN ID-C1
  ID-R2 DELIMITER IN ID-D2
  ID-R3 DELIMITER IN ID-D3 COUNT IN ID-C3
  ID-R4 COUNT IN ID-C4
  WITH POINTER ID-P
  TALLYING IN ID-T
  ON OVERFLOW GO TO OFLOW-EXIT.
```

(All the data receiving fields are defined as alphanumeric)



ID-P (pointer) ID-T (tallying field)

2	1	0	5
---	---	---	---

(after execution-both initialized to 01 before execution)

The order of execution is:

- 1 3 characters are placed in ID-R1.
- 2 Because ALL * is specified, one * is placed in ID-D1.
- 3 5 characters are placed in ID-R2.
- 4 A ? is placed in ID-D2. The current receiving field is now ID-R3.
- 5 A ? is placed in ID-D3; ID-R3 is filled with spaces; no characters are transferred, so 0 is placed in ID-C3.
- 6 No delimiter is encountered before 5 characters fill ID-R4; 8 is placed in ID-C4, representing the number of characters examined since the last delimiter.
- 7 ID-P is updated to 21, the total length of the sending field + 1; ID-T is updated to 5, the number of fields acted upon. Since there are no unexamined characters in the ID-SEND, the OVERFLOW EXIT is not taken.

PART 3. File I/O, Source Control, and Inter-Program Communication

Chapter 8. File Input and Output

Contents

About This Chapter	8-5
Introduction	8-6
Sequential Input-Output	8-6
Relative Input-Output	8-6
Indexed Input-Output	8-7
File Position Indicator	8-7
I-O Status	8-8
AT END Condition	8-11
INVALID KEY Condition	8-11
LINAGE-COUNTER	8-12
Sharing Files on Multiuser Systems	8-12
Exclusive Mode	8-13
Shareable Mode	8-13
Environment Division for File Input and Output	8-16
INPUT-OUTPUT SECTION	8-16
General Format	8-16
FILE-CONTROL Paragraph	8-17
Function	8-17
General Format	8-17
FILE-CONTROL Entry	8-18
Function	8-18
General Formats	8-18
Syntax Rules	8-22
General Rules	8-24
Examples	8-27
I-O Control Paragraph	8-29
Function	8-29
General Format	8-29
Syntax Rules	8-32
General Rules	8-33
Data Division for File Input and Output	8-34
FILE SECTION	8-34
Record Description Structure	8-34
File Description — Complete Entry Skeleton	8-35
BLOCK CONTAINS Clause	8-39
Function	8-39
General Format	8-39
General Rule	8-39
CODE-SET Clause	8-40
Function	8-40
General Format	8-40
Syntax Rules	8-40
DATA RECORDS Clause	8-42
Function	8-42
General Format	8-42
Syntax Rule	8-42
General Rules	8-42
LABEL RECORDS Clause	8-43
Function	8-43
General Format	8-43
Syntax Rules	8-43
General Rule	8-43
LINAGE Clause	8-44
Function	8-44
General Format	8-44
Syntax Rules	8-44
General Rules	8-44
RECORD Clause	8-47
Function	8-47

General Format	8-47
Syntax Rules	8-47
General Rules	8-48
RECORDING MODE Clause	8-50
Function	8-50
General Format	8-50
General Rules	8-50
VALUE OF Clause	8-51
Function	8-51
General Format	8-51
Syntax Rules	8-51
General Rules	8-52
Procedure Division for File Input and Output	8-53
CLOSE Statement	8-53
Function	8-53
General Format	8-53
Syntax Rules	8-54
General Rules	8-54
COMMIT Statement	8-58
Function	8-58
General Format	8-58
General Rules	8-58
DELETE Statement	8-59
Function	8-59
General Format	8-59
Syntax Rules	8-59
General Rules	8-60
Example	8-61
OPEN Statement	8-62
Function	8-62
General Format	8-62
Syntax Rules	8-63
General Rules	8-64
Example	8-67
READ Statement	8-68
Function	8-68
General Format	8-68
Syntax Rules	8-69
General Rules	8-71
Examples	8-74
REWRITE Statement	8-75
Function	8-75
General Format	8-75
Syntax Rules	8-75
General Rules	8-76
Example	8-78
START Statement	8-79
Function	8-79
General Format	8-79
Syntax Rules	8-82
General Rules	8-82
Examples	8-84
UNLOCK Statement	8-85
Function	8-85
General Format	8-85
General Rules	8-85
USE Statement	8-86
Function	8-86
General Format	8-86
Syntax Rules	8-87
General Rules	8-88
Example	8-88

WRITE Statement	8-89
Function	8-89
General Format	8-89
Syntax Rules	8-90
General Rules	8-91
Example	8-96

About This Chapter

This chapter describes the COBOL Input and Output module and its capability to transfer data to and from files stored on external media, and to control low-volume data that is obtained from or sent to an input-output device.

Introduction

IBM AIX VS COBOL allows for three distinct kinds of input-output (I-O):

1. Sequential
2. Relative
3. Indexed.

IBM AIX VS COBOL accepts the clause that refers to reel and tape devices for compatibility. It is accepted syntactically but has no effect at run time.

Sequential Input-Output

Sequential I-O provides a capability to access records of a file in an established sequence. The sequence is established by writing the records to the file. Sequential I-O also provides for the specification of rerun points and the sharing of memory areas among files.

Organization Of Sequential Files

Sequential files are organized so that each record in the file except the first has a unique predecessor record, and each record except the last has a unique successor record. These predecessor-successor relationships are established by the order of WRITE statements when the file is created. Once established, the predecessor-successor relationships do not change except when records are added to the end of the file.

Sequential files are further divided into two subclasses: record-sequential and line-sequential. Record-sequential files have fixed-length records, whereas line-sequential records are stored with trailing spaces removed and are thus variable in length. If the words sequential file or sequential organization are used in this chapter without specifying LINE or RECORD, the sentence in which they are used applies to both forms. *MF*

Access Mode

The only access available for sequential files is sequential access mode. The sequence in which records are accessed is the order in which the records were originally written.

Relative Input-Output

Relative I-O provides a capability to access records of a mass storage file in either a random or sequential manner. Each record in a relative file is uniquely identified by an integer value greater than zero which specifies the record's ordinal position in the file.

Organization Of Relative Files

Relative file organization is permitted only on fixed-disk devices. A relative file consists of records identified by relative record numbers. The file may be thought of as composed of a serial string of areas, each capable of holding a logical record. Each of these areas is identified by a relative record number. Records are stored and retrieved based on this number. For example, you may retrieve or store the tenth record area, whether or not records have been written in the first through the ninth record areas.

Access Mode

In the sequential access mode, records are accessed in the ascending order of the relative record numbers of all records that currently exist within a file.

In the random access mode, the programmer controls the sequence in which records are accessed. The desired record is accessed by placing its relative record number in a relative key data item.

In the dynamic access mode, the programmer may change at will from sequential access to random access using the appropriate forms of input-output statements.

Indexed Input-Output

Indexed I-O provides a capability to access records of a mass storage file in either a random or sequential manner. Each record in an indexed file is uniquely identified by the value of one or more keys within that record.

Organization of Indexed Files

A file whose organization is indexed is a mass storage file in which data records may be accessed by the value of a key. A record description may include one or more key data items, each of which is associated with an index. Each index provides a logical path to the data records according to the contents of a data item within each record which is the record key for that index.

The data item named in the RECORD KEY clause of the file control entry for a file is the prime record key for that file. For purposes of inserting, updating, and deleting records in a file, each record is identified solely by the value of its prime record key. This value, therefore, must be unique and must not be changed when updating the record. Key lengths must not exceed 127 bytes, and this may be further restricted depending on your indexed sequential file run-time module. Refer to the *User's Guide*.

A data item named in the ALTERNATE RECORD KEY clause of the file control entry for a file is an alternative record key for that file. The value of an alternative record key may not be unique if the DUPLICATES phrase is specified for it. These keys provide alternative access paths for retrieval of records from the file. A maximum number of 63 alternate keys can be specified.

Access Mode

In the sequential access mode, records are accessed in the ascending order of the record key values. The order of retrieval of records within a set of records having duplicate record key values is based on the order in which the records were written into the set.

In the random access mode, the programmer controls the sequence in which records are accessed. The desired record is accessed by placing the value of its record key in the record key data item.

In the dynamic access mode, the programmer may change at will from sequential access to random access using appropriate forms of input-output statements.

File Position Indicator

The file position indicator is a conceptual entity used in this document to specify exactly the next record to be accessed within a file during certain sequences of input-output operations. The setting of the file position indicator is affected only by the OPEN, START, and READ statements. The concept of a file position indicator has no meaning for a file opened in the output or extend mode.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item to indicate to the COBOL program the status of that input-output operation. The value is placed into the two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START statement, before any applicable USE procedure is executed.

Status Key 1

The leftmost character position of the FILE STATUS data item is known as status key 1. Status key 1 is set to indicate one of the following conditions upon completion of the input-output operation on a file of any organization.

A list of indications and their meanings are listed below:

- 0** Successful completion.
Indicates that the input-output statement was successfully executed.
- 1** AT END
Indicates that the sequential READ statement was unsuccessfully executed. This may be a result of an attempt to read a record when no next logical record exists in the file. It may also be a result of the first READ statement being executed for a file described with the OPTIONAL clause and that file was not available to the program at the time its associated OPEN statement was executed.
- 2** Invalid key
Indicates that the input-output statement on a nonsequential file was unsuccessfully executed as a result of one of the following:
 - Sequence error
 - Duplicate key
 - No record found
 - Boundary violation.
- 3** Permanent error
Indicates that the input-output statement was unsuccessfully executed. This may be the result of a boundary violation for a sequential file or the result of an input-output error, such as a data check parity error or a transmission error.
- 4** Logic error
Indicates that the input-output statement was unsuccessfully executed. This may be a result of an improper sequence of input-output operations performed on the file or a result of violating a limit defined by the user.
- 9** Run-time environment error message.
Indicates that the input-output statement was unsuccessfully executed as a result of a condition specified by the run time environment error message number. This value is used only to indicate a condition not indicated by other defined values of status key 1 or by specified combinations of the values of status key 1 and status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operation.

The combination of status key 1 and status key 2 defines the result of the input-output operation as detailed below.

Successful Completion

If status key 1 contains 0 to indicate the successful completion of the input-output operation, status key 2 may contain one of the following values:

- 0 (All files.) No further information is available.
- 2 (Indexed files only.) Indicates one of two possibilities:
 - For a READ statement, the key value for the current key of reference is equal to the value of the same key in the next record within the current key of reference.
 - For a WRITE or REWRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.
- 4 (All files.) Indicates that the length of the record being processed does not conform to the fixed file attributes for that file.
- 5 (All files.) Indicates that the referenced optional file is not present at the time the OPEN statement is executed. The OPEN statement was successful. If the OPEN mode is I-O or extend, the file has been created.
- 7 (Sequential files only.) Indicates that the referenced file is a nonreel/unit medium for a CLOSE statement with the NO REWIND, REEL/UNIT or FOR REMOVAL phrase, or for an OPEN statement with the NO REWIND phrase.

AT END Condition with Unsuccessful Completion

If status key 1 contains 1 to indicate the AT END condition, status key 2 can contain one of the following values:

- 0 (All files.) Indicates there is no next logical record. This can be caused by two conditions:
 - The end of the file has been reached.
 - A sequential READ statement is attempted for the first time on an optional input file that is not present.
- 4 (Relative files only.) Indicates the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

INVALID KEY Condition with Unsuccessful Completion

If status key 1 contains 2 to indicate INVALID KEY condition, status key 2 indicates the cause of the condition by one of the following values:

- 1 (Sequentially accessed indexed files.) Indicates a sequence error. The ascending sequence requirements of successive record key values were violated (see "WRITE Statement" on page 8-89), or the prime record key value was changed by the COBOL program between the successful execution of a READ statement and the execution of the next REWRITE statement for the file.
- 2 (Relative and indexed files only.) Indicates a duplicate key value. An attempt is made to either WRITE a record that would create a duplicate prime record key or to WRITE or REWRITE a record that would create a duplicate alternate record key without the DUPLICATES phrase.

-
- 3 Indicates no record found. It may be an attempt was made to access a record, identified by a key, and that record does not exist in the file. It may also be that a START or READ statement was attempted on an optional input file that is not present.
 - 4 (Relative and indexed files only.) Indicates a boundary violation arising from one of the following conditions:
 - An attempt is made to write beyond the externally defined boundaries of a file.
 - A sequential WRITE statement is attempted for a relative file, but the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

Permanent Error Condition with Unsuccessful Completion

If status key 1 contains 3 to indicate a permanent error condition, status key 2 can contain one of the following values to indicate the cause of the error:

- 0 (All files.) Indicates that no further information is available concerning the cause of the error.
- 4 (Sequential files only.) Indicates a boundary violation. An attempt was made to write beyond the externally defined boundaries of a file.
- 5 (All files.) Indicates that an OPEN statement with the INPUT, I-O, or EXTEND phrase was attempted on a required file that is not present.
- 7 (All files.) Indicates that an OPEN statement was attempted on a file which will not support the open mode specified in the OPEN statement. The possible violations are:
 - The EXTEND or OUTPUT phrase is specified but the file does not support write operations.
 - The I-O phrase is specified but the file does not support the input and output operations permitted for a relative file when opened in the I-O mode.
 - The INPUT phrase was specified but the file does not support read operations.
- 8 (All files.) Indicates that an OPEN statement was attempted on a file previously closed with lock.
- 9 (All files.) Indicates that an OPEN statement was unsuccessful because a conflict was detected between the fixed file attributes and the attributes specified for that file in the program.

Logic Error Condition with Unsuccessful Completion

If status key 1 contains 4 to indicate a logic error condition, status key 2 can contain one of the following values to indicate the cause of the error:

- 1 (All files.) Indicates that an OPEN statement was attempted for a file already in the open mode.
- 2 (All files.) Indicates that a CLOSE statement was attempted for a file not in the open mode.
- 3 (All files, sequential access mode only.) Indicates that the last input-output statement executed for the associated file, prior to the execution of a DELETE or REWRITE statement, was not a successfully executed READ statement.

-
- 4 (All files.) Indicates that a boundary violation exists. Possible violations are:
- An attempt was made to WRITE or REWRITE a record larger than the largest record allowed or smaller than the smallest record allowed. The largest and smallest records allowed are specified in the RECORD IS VARYING clause of the associated file.
 - An attempt was made to REWRITE a record to a file, and the record is not the same size as the record being replaced.
- 6 (All files.) Indicates that a sequential READ statement was attempted on a file open in the input or I-O mode but no valid next record was established. This can be caused by the following conditions:
- The preceding START statement was unsuccessful.
 - The preceding READ statement was unsuccessful but did not cause an END condition.
 - The preceding READ statement caused an AT END condition.
- 7 (All files.) Indicates that the execution of a READ or START statement was attempted on a file not open in the input or I-O mode.
- 8 (All files.) Indicates that the execution of a WRITE statement was attempted on a file not open in the I-O, output, or extend mode.
- 9 (All files.) Indicates that the execution of a DELETE or REWRITE statement was attempted on a file not open in the I-O mode.

Run Time Environment Error Message

If status key 1 contains 9 to indicate a run time environment error message, status key 2 contains the run time environment error message number. Refer to the *User's Guide*.

AT END Condition

The AT END condition can occur as a result of the execution of a READ statement. Refer to "READ Statement" on page 8-68 for details of the causes of the condition.

INVALID KEY Condition

The INVALID KEY condition can occur as a result of the execution of a START, READ, WRITE, REWRITE, or DELETE statement. Refer to "START Statement" on page 8-79, "READ Statement" on page 8-68, "WRITE Statement" on page 8-89, "REWRITE Statement" on page 8-75, and "DELETE Statement" on page 8-59 for details on the causes of the condition.

When the INVALID KEY condition is recognized, the run time environment takes these actions in the following order:

1. A value is placed in the FILE STATUS data item, if specified for this file, to indicate an INVALID KEY condition. Refer to "I-O Status" on page 8-8.
2. If the INVALID KEY phrase is specified in the statement causing the condition, control is passed to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.
3. If the INVALID KEY phrase is not specified, but a USE procedure is specified for this file, either explicitly or implicitly, the procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement which recognized the condition is unsuccessful and the file is not affected.

Note: INVALID KEY does not trap errors when status key 1 is set to 9. Such errors must be trapped either by explicitly testing the status key or by using declaratives instead of the INVALID KEY clause.

Valid Combinations of Status Keys 1 and 2

In Figure 8-1, S indicates a sequential file, R indicates a relative file, and I indicates an indexed file.

A particular combination of status key 1 and status key 2 is valid for a given file organization if the letter for the organization is found at the corresponding intersection in the table.

Status Key 1	Status Key 2									
	0	1	2	3	4	5	6	7	8	9
Successful Completion, 0	SRI		I		SRI	SRI		S		
At End, 1	SRI				R					
Invalid Key, 2		I	RI	RI	RI					
Permanent Error, 3	SRI				S	S		SRI	SRI	SRI
Logic Error, 4	SRI		SRI	SRI	SRI		SRI	SRI	SRI	SRI
Implementor Defined, 9	Run Time Environment Error Message (SRI)									

Figure 8-1. Valid Combinations of File Status Keys 1 and 2

LINAGE-COUNTER

The reserved word LINAGE-COUNTER is a name for a special register generated by the presence of a LINAGE clause in a file description entry for a sequential file. The implicit description is that of an unsigned integer whose size is equal to integer-1 or the data item referenced by data-name-1 in the LINAGE clause. Refer to "LINAGE Clause" on page 8-44.

Sharing Files on Multiuser Systems

The Run Time Environment (RTE) supports the AIX VS COBOL multiuser facilities. The multiuser facilities allow files to be shared between users in a multiuser environment and allow programs accessing those files to prevent access to records or entire files while data is being updated.

Files are either active or inactive. An active file is open to one or more run-units. An inactive file is not open to any run-unit.

Active files may be in one of two modes:

- exclusive
- shareable.

Exclusive Mode

A file in exclusive mode is open to one run-unit only, and any other run-unit which attempts to access it receives a file locked error and is denied access. Exclusive mode implies that a file lock is held by the one run-unit which is able to access the file; the file lock is released by the run-unit closing the file.

Shareable Mode

A file in shareable mode is available to any number of run-units, each of which may protect data while using the file by locking one or more records in the file. This prevents other run-units accessing the individual locked records, but does not prevent access to the file otherwise. The organization of the file affects the way the file can be shared.

Sequential Files

Line-sequential files opened in OUTPUT or EXTEND mode can explicitly or implicitly be made shareable, but records cannot be locked in the file. A run-unit cannot lock multiple records in a file whose organization is sequential. Each run-unit sharing access to a sequential file may be locking a single record in that file.

Relative and Indexed Files

Files opened in INPUT mode are shareable, but records cannot be locked in the file. Each run-unit sharing access to a file may be locking either a single record or multiple records in that file. A run-unit cannot select single record locking and multiple record locking for the same file.

Single Record Lock

A run-unit that has specified single record locking for a file (either explicitly or implicitly) can hold only one record lock in that file at any time. There are two ways for a run-unit to acquire a record lock:

- Manually
- Automatically.

Manual Record Locking

The run-unit acquires a lock only if it accesses a record with a READ WITH LOCK statement.

For sequential files, the lock is released by the same run-unit:

- Accessing any record in the file with any file operation
- Executing an UNLOCK statement on that file
- Executing a COMMIT statement
- Closing the file.

For relative and indexed files, the lock is released by the same run-unit:

- Accessing any record in the file with any file operation except START
- Executing an UNLOCK statement on the file
- Executing a COMMIT statement on the file
- Closing the file.

Automatic Record Locking

The run-unit acquires a lock whenever it reads a record in the file. The lock is released as for manual record locking. Refer to the previous section.

Multiple Record Locks

Locking of multiple records in a file is available only when the organization of that file is relative or indexed.

A run-unit that has specified multiple record locking for a file may hold a number of record locks in one file simultaneously. This prevents other run-units accessing those locked records but does not deny them access to any records that are not locked. There are two ways record locks can be acquired: manually or automatically.

Manual Record Locking

The run-unit acquires a lock only if it accesses a record with a **READ WITH KEPT LOCK** statement. If the **WRITELOCK** directive was specified at the time the program was compiled, then a lock is acquired if the run-unit accesses the file with a **WRITE** or **REWRITE** statement. The locks are released by the same run-unit:

- Executing an **UNLOCK** statement for that file
- Executing a **COMMIT** statement
- Closing the file.

Automatic Record Locking

The run-unit acquires a lock whenever it reads a record in the file. If the **WRITELOCK** directive is specified at the time the program is compiled, then a lock is acquired if the run-unit accesses the file with a **WRITE** or **REWRITE** statement. The locks are released by the same run-unit:

- Executing an **UNLOCK** statement for that file
- Executing a **COMMIT** statement
- Closing the file.

Table 8-1 and Table 8-2 show the default type of locking used when files are opened in a particular open mode. The default locking can be modified if the **AUTOLOCK** directive is specified at the time the program is compiled. The table also indicates whether the default type of locking may be overridden for individual files. This is done by inserting a suitable clause in the **SELECT** statement for the file.

OPEN Mode	No Directive	AUTOLOCK Directive	Override in SELECT Statement
INPUT	No lock	No lock	Yes, but only to EXCLUSIVE .
I-O	Exclusive	Automatic	Yes, lock on single record.
OUTPUT	Exclusive	Exclusive	No.
EXTEND	Exclusive	No lock	Yes, the file can be made shareable but no records are locked.

Table 8-2. Default Locking for Relative and Indexed Files			
OPEN Mode	No Directive	AUTOLOCK Directive	Override in SELECT Statement
INPUT	No lock	No lock	Yes, but only to EXCLUSIVE.
I-O	Exclusive	Automatic lock on single record	Yes.
OUTPUT	Exclusive	Exclusive	No.

Notes:

1. A file opened for OUTPUT causes the file to become exclusive, regardless of the specified lock mode.
2. Explicitly or implicitly specifying automatic or manual record locking for a file causes the file to become shareable. A file opened for I-O acquires record locks. A file opened for INPUT or EXTEND never acquires record locks.
3. A programmer can select the type of locking for individual files by accepting the default locking or by including a LOCK MODE clause in the file control entry. Refer to "FILE-CONTROL Entry" on page 8-18.

In single user environments the multiuser syntax has no effect on run time, but programs can be developed for use in both single and multiuser environments.

Environment Division for File Input and Output

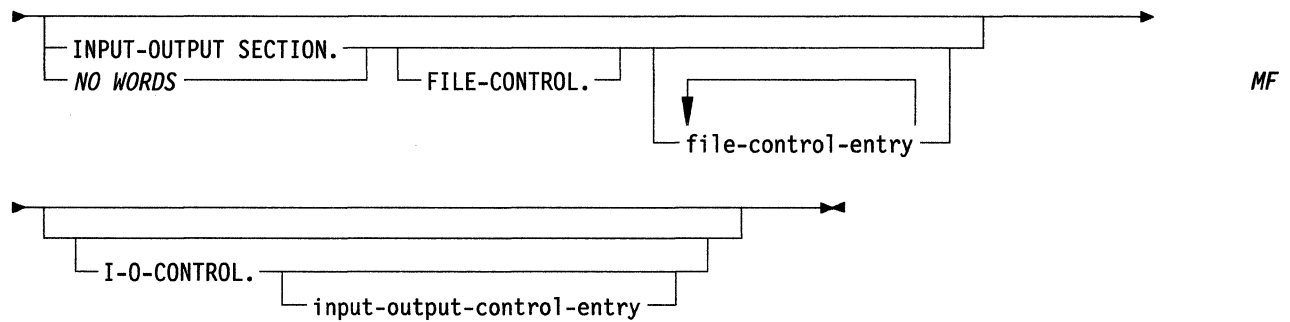
File input and output for the Environment Division of the program is described in the following sections.

INPUT-OUTPUT SECTION

File input and output for the Environment Division is controlled by the INPUT-OUTPUT SECTION of the program.

General Format

The following figure shows the general format of the Input-Output Section:



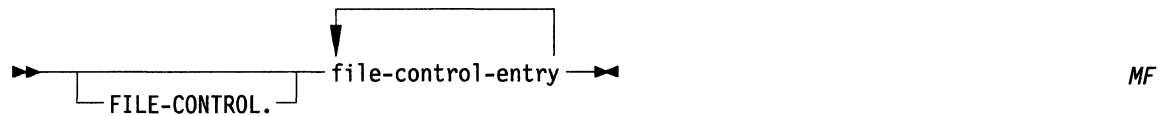
FILE-CONTROL Paragraph

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

General Format

The following figure shows the general format of the FILE-CONTROL paragraph:



The FILE-CONTROL reserved word is optional when the MF option is used.

FILE-CONTROL Entry

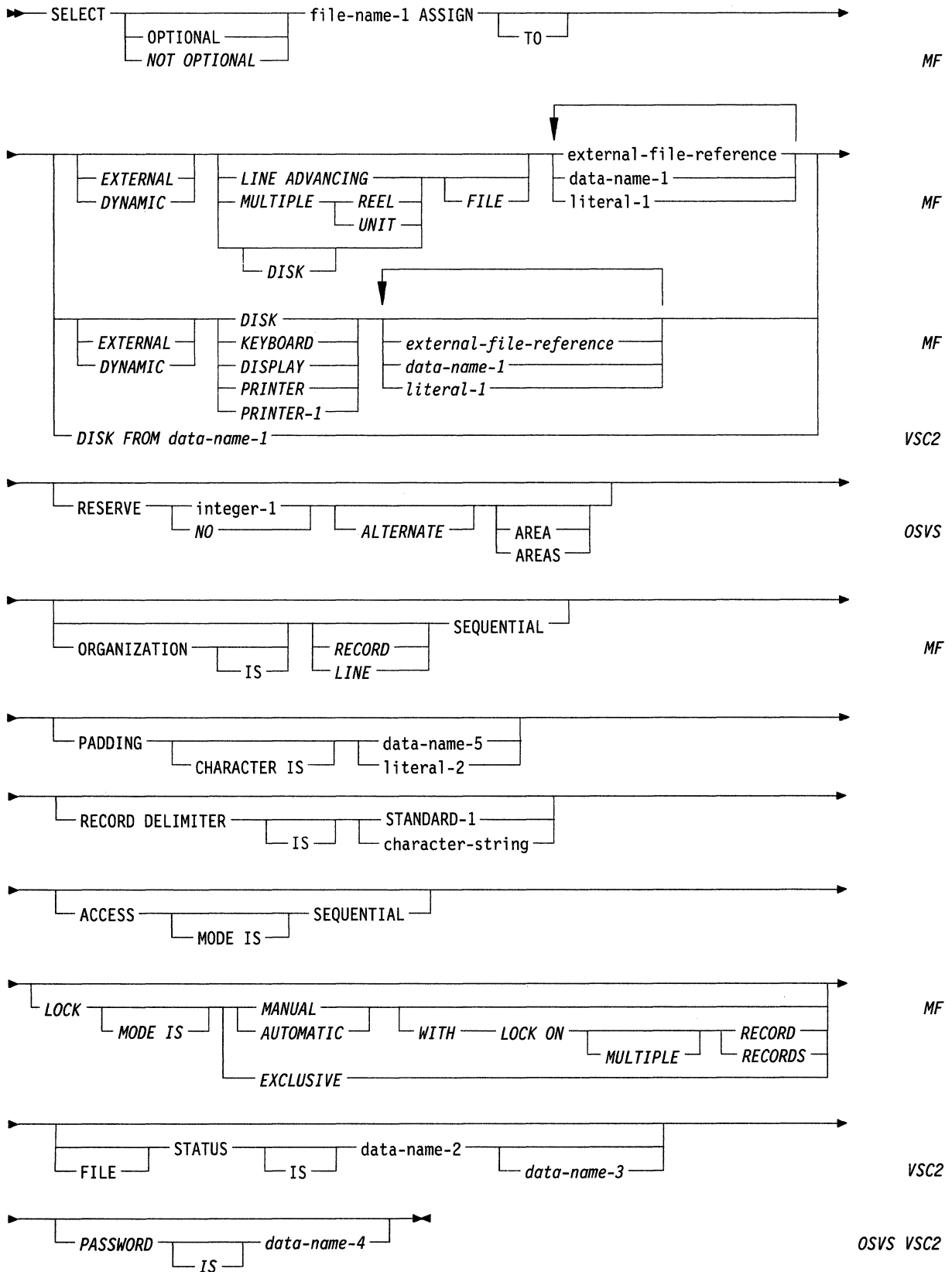
Function

The FILE-CONTROL entry names a file and may specify other file-related information.

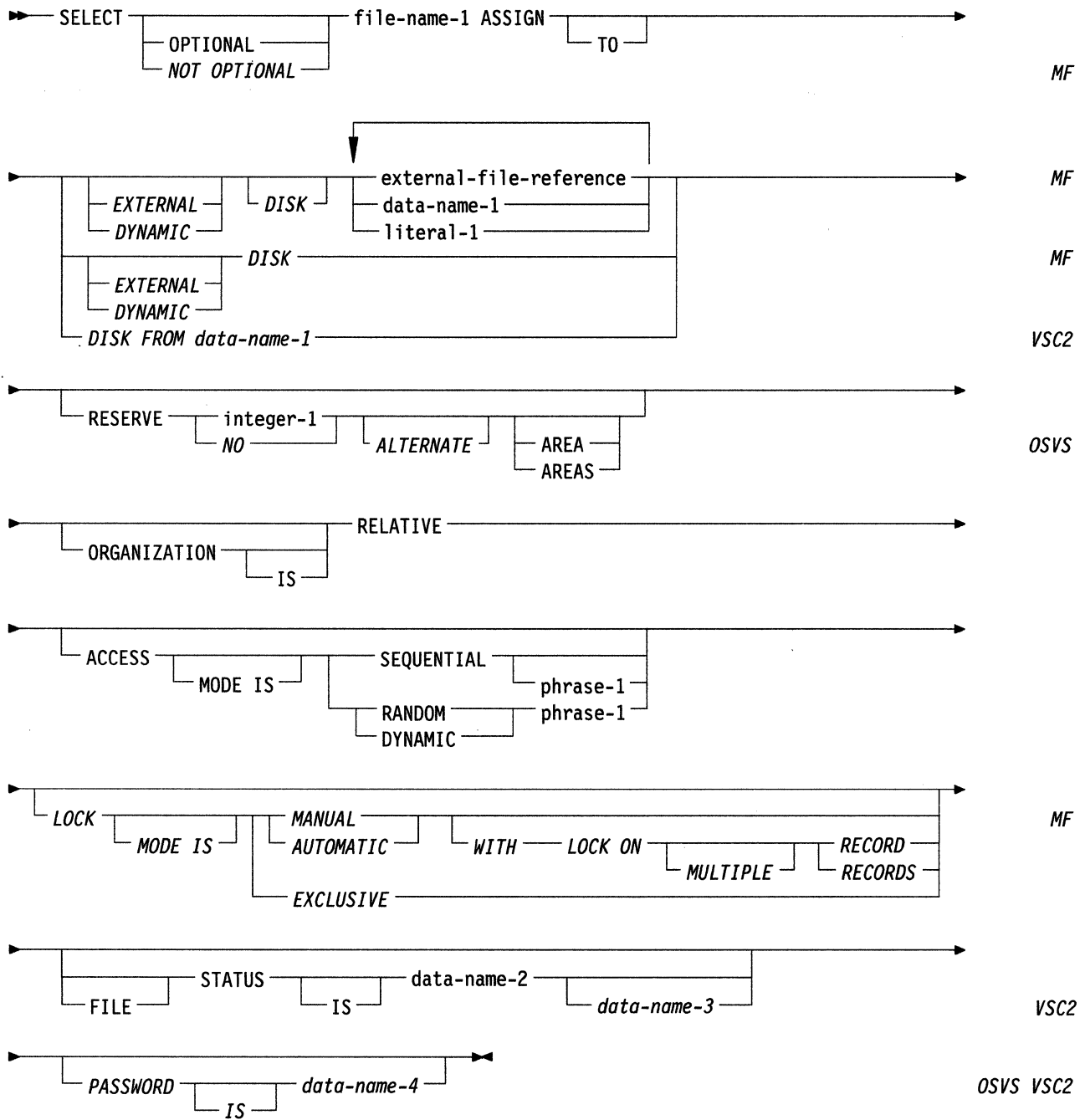
General Formats

The following figures show the general formats of the FILE-CONTROL entry:

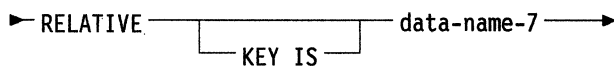
Format 1 (Sequential Files)



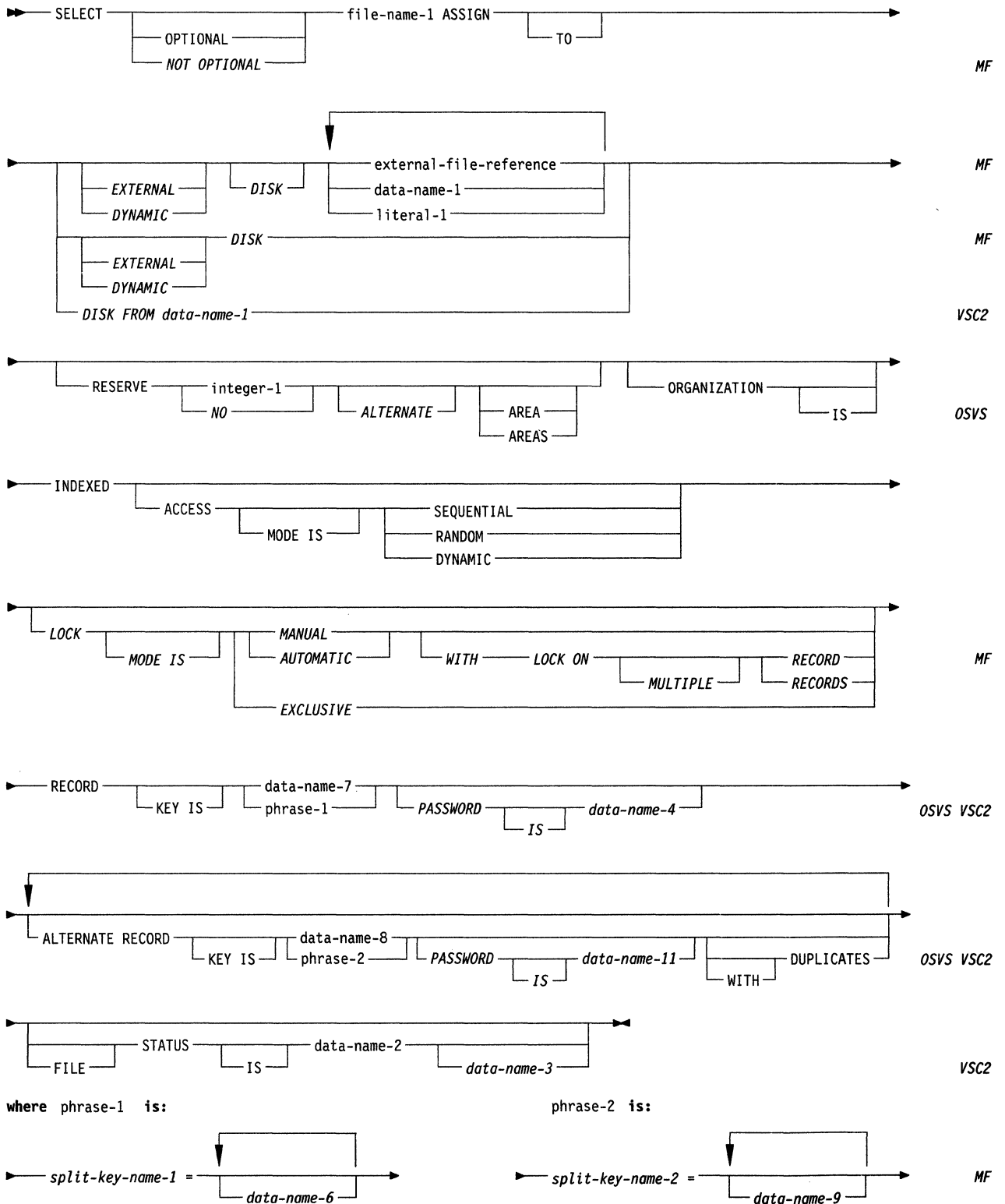
Format 2 (Relative Files)



where phrase-1 is:



Format 3 (Indexed Files)



Syntax Rules

The following rules apply to the FILE-CONTROL entry:

All Formats (All Files)

1. The SELECT clause must be specified first in the file control entry. The clauses following the SELECT clause may appear in any order.
2. Each file described in the Data Division must be named once, and only once, as file name in the FILE-CONTROL paragraph. Each file specified in the file control entry must have a file description entry in the Data Division.
3. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.
4. data-name-2 must be defined in the Data Division as a two-character alphanumeric data item, or as a two-character numeric data item with USAGE DISPLAY. data-name-2 must not be defined in the FILE SECTION, the REPORT SECTION, or the COMMUNICATION SECTION. OSVS VSC2
5. data-name-3 must be defined as a group item of 6 bytes in the WORKING-STORAGE or LINKAGE SECTION of the Data Division. VSC2
6. All data-names may be qualified.
7. data-name-1 may be declared in the Data Division as an alphanumeric or group data item long enough to hold the external name of the file. If it is not explicitly declared within the program, the AIX VS COBOL system will declare it implicitly as an alphanumeric data item long enough to hold the maximum permissible size of program name. MF
If data-name-1 occurs in the FROM option, the data item must be explicitly declared in the Data Division. VSC2
8. The NOT OPTIONAL phrase may be specified only for files to be opened input-output. MF
9. If both EXTERNAL and DYNAMIC are omitted from the ASSIGN clause, one of these options is assumed according to the setting of a COBOL system directive. Refer to the User's Guide. MF
If the EXTERNAL option is used, then an external-file reference must be specified.
If the DYNAMIC option is used, then data-name-1 or literal-1 must be specified.
10. data-name-4 and data-name-11 must be defined in the WORKING-STORAGE SECTION as an alphanumeric data item. OSVS VSC2
11. The OPTIONAL phrase may only be specified for files opened in the INPUT, I-O, or EXTEND mode. Its specification is required for files that are not always present each time the object program is executed.

Format 1 (Sequential Files)

12. KEYBOARD means console input. MF
13. DISPLAY means console output. MF
14. PRINTER specifies the main printer on the system. MF
15. PRINTER-1 specifies the second printer on the system. MF
16. If both RECORD and LINE are omitted from the ORGANIZATION phrase, one of these options is assumed according to the setting of the COBOL system directive SEQUENTIAL. Refer to the User's Guide. MF

-
17. When the ORGANIZATION clause is not specified, sequential organization is assumed. The exact format (LINE-SEQUENTIAL or RECORD-SEQUENTIAL) depends on the setting of the AIX VS COBOL system directive SEQUENTIAL. Refer to the *User's Guide*.
 18. literal-2 must be a one character nonnumeric literal.
 19. data-name-5 may be qualified. It must be defined in the Data Division as a one character data item of the alphanumeric category. data-name-5 cannot be defined in the COMMUNICATION, FILE, or REPORT SECTIONS of the Data Division.
 20. Character string must not be a reserved word, a user name or a literal.

Format 2 (Relative Files)

21. If a relative file is to be referenced by a START statement, the RELATIVE KEY phrase must be specified for that file.
22. data-name-7 must not be defined in a record description entry associated with that file name.
23. The data item referenced by data-name-7 must be defined as an unsigned integer.

Format 3 (Indexed Files)

24. *split-key is a concatenation of one or more data items within a record associated with the file name. It can be referenced only in START and READ statements.* MF
25. The data items referenced by data-name-7 and data-name-8 and any data-names referenced by split-key-name-1 and split-key-name-2 must be defined as alphanumeric data items within a record description entry associated with the file name. MF
26. data-name-7, data-name-8, and any data-names referenced by split-key-name-1 and split-key-name-2 cannot describe an item whose size is variable. Refer to "OCCURS Clause" on page 12-5. MF
27. data-name-8 cannot reference an item whose leftmost character position corresponds to the leftmost character position of an item referenced by data-name-7 or by any other data-name associated with this file.
28. *If the PASSWORD clause is specified, it must immediately follow its associated RECORD KEY or ALTERNATE RECORD KEY clause.* OSVS VSC2

DBCS Support

29. data-name-7 and data-name-8 can be DBCS data items. The key is treated as an alphanumeric item for the input and output statements for the file names in the SELECT clause. When you specify data-name-7 or data-name-8 as a DBCS data item, a KEY specified on the READ statement must also be a DBCS data item.

End of DBCS Support

General Rules

The following rules apply to the file-control entry:

All Formats (All Files)

1. The ASSIGN clause specifies the association of the file referenced by file name to a storage medium. Refer to the *User's Guide*. The first assignment takes effect. Subsequent assignments within any one ASSIGN clause are for documentation purposes only.
2. The RESERVE clause allows the user to specify the number of input-output areas allocated. The RESERVE clause is for documentation purposes only. However, it does have the effect of implicitly declaring a data item AREA-VALUE, which is PIC 9(2) COMP and contains the number of areas reserved.
3. The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot be changed.
4. When the FILE STATUS clause is specified, a value is moved by the Run Time Environment into the data item specified by data-name-2 after the execution of every statement that references the file either explicitly or implicitly. This value indicates the status of execution of the statement. Refer to "I-O Status" on page 8-8.
5. *Data-name-3, if specified, is documentary only.* VSC2
6. *The PASSWORD clause is documentary only.* OSVS VSC2
7. *The use of the reserved word DYNAMIC in an ASSIGN clause indicates that the value of the literal-1 or data-name-1 in the ASSIGN clause is the name of the specified file in the external file storage environment.* MF
8. *The use of the reserved word EXTERNAL in an ASSIGN clause indicates that external-file-reference identifies the specified file to the external environment for possible further mapping to an external file storage environment name. Refer to the User's Guide for details on setting up the external file name for your particular operating environment.* MF
If the external-file-reference contains the character -, then only the part of the name following the last - is used to identify the file to the external environment.
9. *The DISK option without external file reference, data-name-1, or literal-1 specifies a disk file whose name on the fixed-disk will be given in a VALUE OF FILE-ID clause in the file description of the file. If the file description contains no VALUE OF FILE-ID clause, the name of the disk file is assumed to be the same as the internal file name.* MF
10. *If any of the keywords DISK, KEYBOARD, DISPLAY, PRINTER, or PRINTER-1 is followed by external-file-reference, data-name-1, or literal-1, the keyword is ignored.* MF
Otherwise the defaults used are:
KEYBOARD "stdin"
DISPLAY "stdout"
PRINTER "|dev/lp0"
PRINTER-1 "|dev/lp1"
DISK file-name-1
11. *The DISK option with the FROM option specifies a disk file whose name on the fixed-disk is the value of data-name-1. However, if an OPEN statement is executed for that file, and data-name-1 contains all spaces, the name of the disk file is assumed to be the same as the internal file name.* VSC2

-
12. *The LOCK MODE clause is an optional clause of the file control entry and is used to specify the locking technique used for the file.* **MF**

If the LOCK MODE clause is omitted, opening the file causes it to become exclusive, unless the file is opened for INPUT. In that case, the file becomes shareable.

When LOCK MODE IS EXCLUSIVE is specified, the run-unit acquires a lock on the whole file when it opens the file.

When LOCK MODE IS AUTOMATIC or LOCK MODE IS MANUAL is specified, the file the run-unit opens is shareable.

The LOCK MODE clause is documentary for line sequential files.

Format 1 (Sequential Files)

13. The MULTIPLE REEL or MULTIPLE UNIT phrase should be specified if it is possible or intended for the file to be closed by use of the CLOSE REEL or CLOSE UNIT statement. Use of the MULTIPLE REEL or MULTIPLE UNIT phrase causes an error if the FILESHARE directive was specified on the AIX VS COBOL command line.

Magnetic tape access is allowed to blocked devices only. Multiple reel and rewind syntax is accepted by the compiler, but has no effect at run time.

14. When LINE-SEQUENTIAL ORGANIZATION is specified, either implicitly or explicitly, the file is treated as though it had variable-length records. Each record contains one line of data. The definition of a line of data varies with different operating systems. Some terminate line records with the Carriage Return and Line Feed characters, or one of them, and some pad the records out as fixed-length records. Therefore, AIX VS COBOL is always compatible with the editor software in any operating system in this respect.
15. Records in the file are accessed in the sequence specified by predecessor-successor record relationships. Predecessor-successor record relationships are established by the execution of WRITE statements when the file is created or extended.
16. Using the LINE ADVANCING FILE phrase causes a file suitable for a printer to be produced. This file will have no space compression for tab characters. The file will have an initial carriage-return (0D hexadecimal) character. Each record in the file is written with AFTER ADVANCING 1 LINE as the default advancing phrase.
17. *The WITH LOCK ON RECORD clause is for documentary purposes only. LOCK MODE IS MANUAL and LOCK MODE IS AUTOMATIC imply single record locking.* **MF**
- Line-sequential files cannot be opened for I-O and so cannot acquire a record lock.*
18. *The PADDING CHARACTER clause is for documentary purposes only.* **MF**
19. *The RECORD DELIMITER clause is for documentary purposes only.* **MF**

Format 2 (Relative Files)

20. When the access mode is sequential, records in the file are accessed in order of ascending relative record numbers of existing records in the file.
21. If the access mode is random, the value of the RELATIVE KEY data item indicates the record to be accessed.
22. When the access mode is dynamic, records in the file may be accessed sequentially and/or randomly. Refer to general rules 20 and 21 in this list.
23. All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a given record specifies the logical ordinal position of the record in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4,....
24. The data item specified by data-name-7 is used to communicate a relative record number between the user and the operating system.

-
25. The *WITH LOCK ON RECORD* clause specifies single record locking for the file.

MF

The WITH LOCK ON MULTIPLE RECORDS clause specifies multiple record locking for the file. This clause must be present if multiple record locking is required.

If LOCK MODE IS AUTOMATIC WITH LOCK ON RECORD is specified for a file, a record lock is acquired by the execution of the READ statement and released on the execution of a subsequent I-O operation, with the exception of a START statement.

If LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORDS is specified for a file, a record lock is acquired by the execution of the READ statement and is not released until a CLOSE, COMMIT, or UNLOCK statement is executed. Refer to the User's Guide.

If the LOCK MODE IS MANUAL WITH LOCK ON RECORD is specified, a lock is acquired by a READ statement only when the READ statement includes the WITH LOCK phrase.

If LOCK MODE IS MANUAL WITH LOCK ON MULTIPLE RECORDS is specified, a record lock is acquired by the READ WITH KEPT LOCK statement.

When the lock mode is MANUAL or AUTOMATIC, single record locking is assumed unless the WITH LOCK ON MULTIPLE RECORDS phrase is included.

Format 3 (Indexed Files)

26. When access mode is sequential, records in the file are accessed in order of ascending record key values within a given key of reference.
27. If the access mode is random, the value of the record key data item indicates the record to be accessed.
28. When the access mode is dynamic, records in the file may be accessed sequentially and/or randomly. Refer to rules 26 and 27 in this list.
29. The RECORD KEY clause specifies the prime record key for the file. The values of the prime record key must be unique among records of the file. This prime record key provides an access path to records in an indexed file.
30. An ALTERNATE RECORD KEY clause specifies a record key that is an alternative record key for the file. This alternate record key provides an alternate access path to records in an indexed file.
31. The data description of data-name-7 and data-name-8, as well as relative locations within a record, must be the same as that used when the file was created. The number of alternate keys for the file must also be the same as that used when the file was created.
32. The DUPLICATES phrase specifies that the value of the associated alternate record key may be duplicated within any of the records in the file. If the DUPLICATES phrase is not specified, the value of the associated alternate record key must not be duplicated among any of the records in the file.

33. The *WITH LOCK ON RECORD* clause specifies single record locking for the file.

MF

The *WITH LOCK ON MULTIPLE RECORDS* clause specifies multiple record locking for the file. This clause must be present if multiple record locking is required.

If *LOCK MODE IS AUTOMATIC WITH LOCK ON RECORD* is specified for a file, a record lock is acquired by the execution of the *READ* statement.

If *LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORDS* is specified for a file, a record lock is acquired by the *READ* statement.

If the *LOCK MODE IS MANUAL WITH LOCK ON RECORD* is specified, a lock is acquired by a *READ* statement only if the *READ* statement includes the *WITH LOCK* phrase.

If *LOCK MODE IS MANUAL WITH LOCK ON MULTIPLE RECORDS* is specified, a record lock is acquired by the *READ WITH KEPT LOCK* statement.

When the lock mode is *MANUAL* or *AUTOMATIC*, single record locking is assumed unless the *WITH LOCK ON MULTIPLE RECORDS* phrase is included.

Examples

The following examples show the *FILE-CONTROL* paragraph:

```
ENVIRONMENT DIVISION.  
:  
:  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT ACCT-TRANS ASSIGN TO "ACCTPAY.TRN"  
    RESERVE 2 AREAS  
    ORGANIZATION IS SEQUENTIAL  
    RECORD DELIMITER IS STANDARD  
    ACCESS MODE IS SEQUENTIAL  
    FILE STATUS IS FILE-STATUS.
```

```
ENVIRONMENT DIVISION.  
:  
:  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT ACCT-PAYABLE ASSIGN TO "ACCTPAY.MAS"  
    ORGANIZATION IS RELATIVE  
    ACCESS MODE IS RANDOM  
    RELATIVE KEY IS ACCT-NO  
    FILE STATUS IS FILE-STATUS.
```

ENVIRONMENT DIVISION.

:

:

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT ACCT-HIST ASSIGN TO "ACCTPAY.HIS"
ORGANIZATION IS INDEXED
ACCESS MODE IS DYNAMIC
RECORD KEY IS ACCT-NO
ALTERNATE RECORD KEY IS ACCT-DATE WITH DUPLICATES
FILE STATUS IS FILE-STATUS.

I-O Control Paragraph

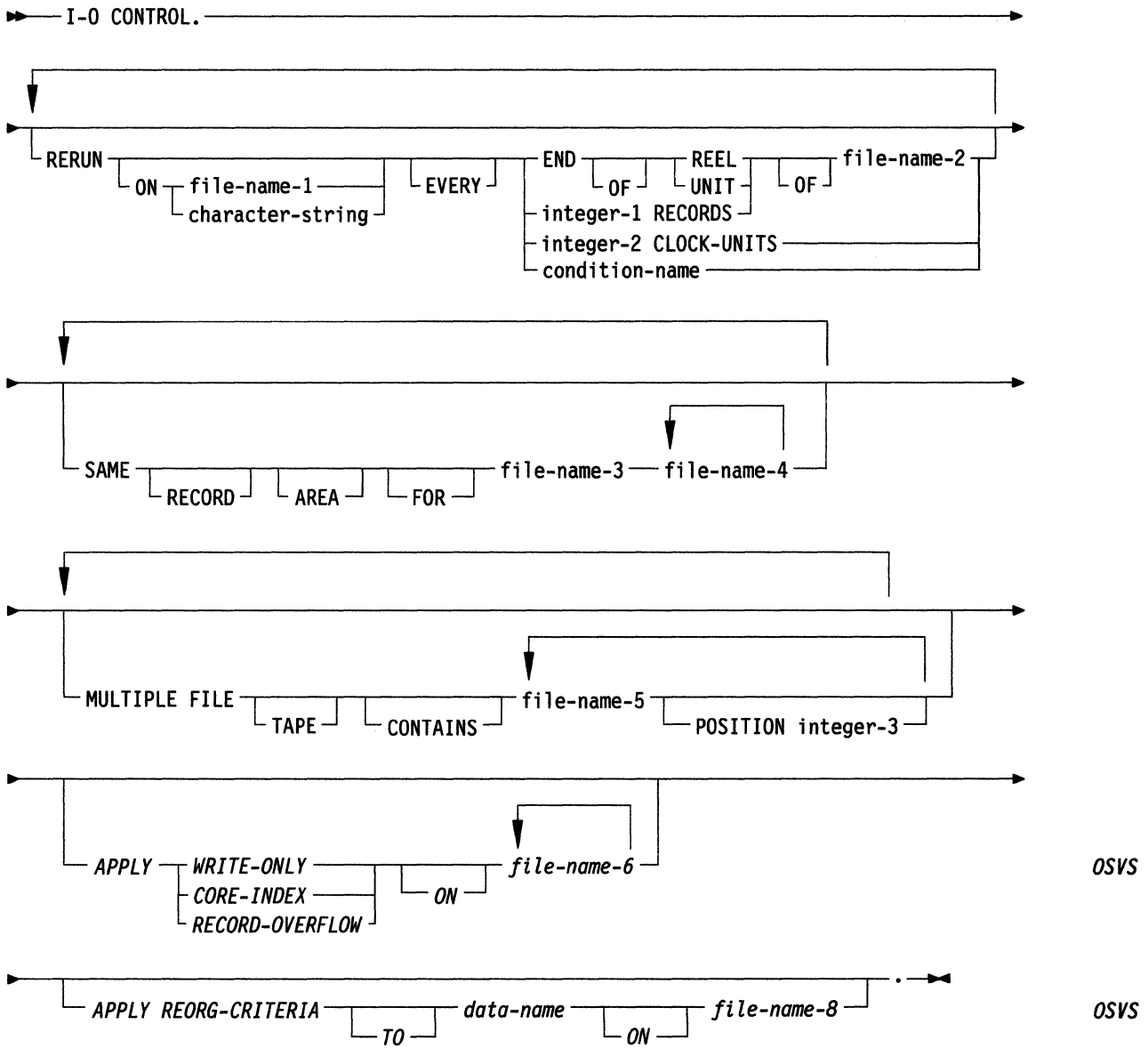
Function

The I-O control paragraph specifies the points at which rerun is established, the memory area shared by different files, and for files with sequential organization, the location of files on a multiple file reel. Multiple file and tape syntax is accepted by the compiler, but has no effect at run time.

General Format

The following figures show the general format of I-O control:

Format 1 (Sequential Files)



Format 2 (Relative and Indexed Files)

► I-O-CONTROL. ◄

┌──┐
└──┘
RERUN ON file-name-1 character-string EVERY integer-1 RECORDS OF file-name-2
integer-2 CLOCK-UNITS
condition-name

┌──┐
└──┘
SAME RECORD AREA FOR file-name-3

┌──┐
└──┘
APPLY WRITE-ONLY CORE-INDEX RECORD-OVERFLOW ON file-name-6 OSVS

┌──┐
└──┘
APPLY REORG-CRITERIA TO data-name ON file-name-8 OSVS

Syntax Rules

This following syntax rules apply to I-O control:

All Formats (All Files)

1. The I-O CONTROL paragraph is optional.
2. file-name-1 must be a sequentially organized file.
3. When either the integer-1 RECORDS clause or the integer-2 CLOCK-UNITS clause is specified, a character-string must be given in the RERUN clause.
4. The two forms of the SAME clause (SAME AREA, SAME RECORD AREA) are considered separately in the following.

More than one SAME clause may be included in a program, subject to the following restrictions:

- a. A file name must not appear in more than one SAME AREA clause.
 - b. A file name must not appear in more than one SAME RECORD AREA clause.
 - c. If one or more file names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file names in that SAME AREA clause must appear in the SAME RECORD AREA clause. However, additional file names not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause. The rule that only one of the files mentioned in a SAME AREA clause can be open at any given time takes precedence over the rule that all files mentioned in a SAME RECORD AREA clause can be open at any given time.
5. *Character-string must not be a reserved word, a literal, or a user name.* *MF*
 6. The files referenced in the SAME AREA or SAME RECORD AREA clause need not all have the same organization or access.

Format 1 (Sequential Files)

7. The END OF REEL/UNIT clause may only be used if file-name-2 is a sequentially organized file.
8. More than one RERUN clause may be specified for a given file-name-2 subject to the following restrictions:
 - a. When multiple integer-1 RECORD clauses are specified, no two of them can specify the same file-name-2.
 - b. When multiple END OF REEL or END OF UNIT clauses are specified, no two of them may specify the same file-name-2.

Format 2 (Relative And Indexed Files)

9. When multiple integer-1 RECORDS clauses are specified, no two of them may specify the same file-name-2.
10. Only one RERUN clause containing the CLOCK-UNITS clause may be specified.

General Rules

The following general rules apply to I-O control:

All Formats (All Files)

1. The RERUN clause is treated as for documentation purposes only.
2. The SAME AREA clause specifies that two or more files that do not represent sort or merge files are to use the same memory area during processing. The area being shared includes all memory area assigned to the files specified. Therefore, it is not valid to have more than one of the files open at the same time. Refer to syntax rule 4c on page 8-32.
3. The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file listed in this SAME RECORD AREA clause and of the most recently read input file listed in this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area; records are aligned on the leftmost character position.
4. *The APPLY clause is for documentation purposes only.* OSVS

Format 1 (Sequential Files)

5. The MULTIPLE FILE clause is for documentation purposes only.

Data Division for File Input and Output

This section describes file input and output for the Data Division of an AIX VS COBOL program.

FILE SECTION

In an AIX VS COBOL program, the file description entry (FD) represents the highest level of organization in the FILE SECTION. The FILE SECTION header is followed by a file description entry consisting of a level indication (FD), a file name, and a series of independent clauses. The FD clauses specify the size of the logical and physical records, the presence or absence of label records, the value of implementer-defined label items, and the names of the data records which comprise the file. The entry itself is terminated by a period.

Record Description Structure

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level number followed by a data-name, if required, followed by a series of independent clauses as required. A record description has a hierarchical structure. Therefore, the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in "Concept Of Levels" on page 2-16, while the elements allowed in a record description are shown in "Data Description — Complete Entry Skeleton" on page 6-7.

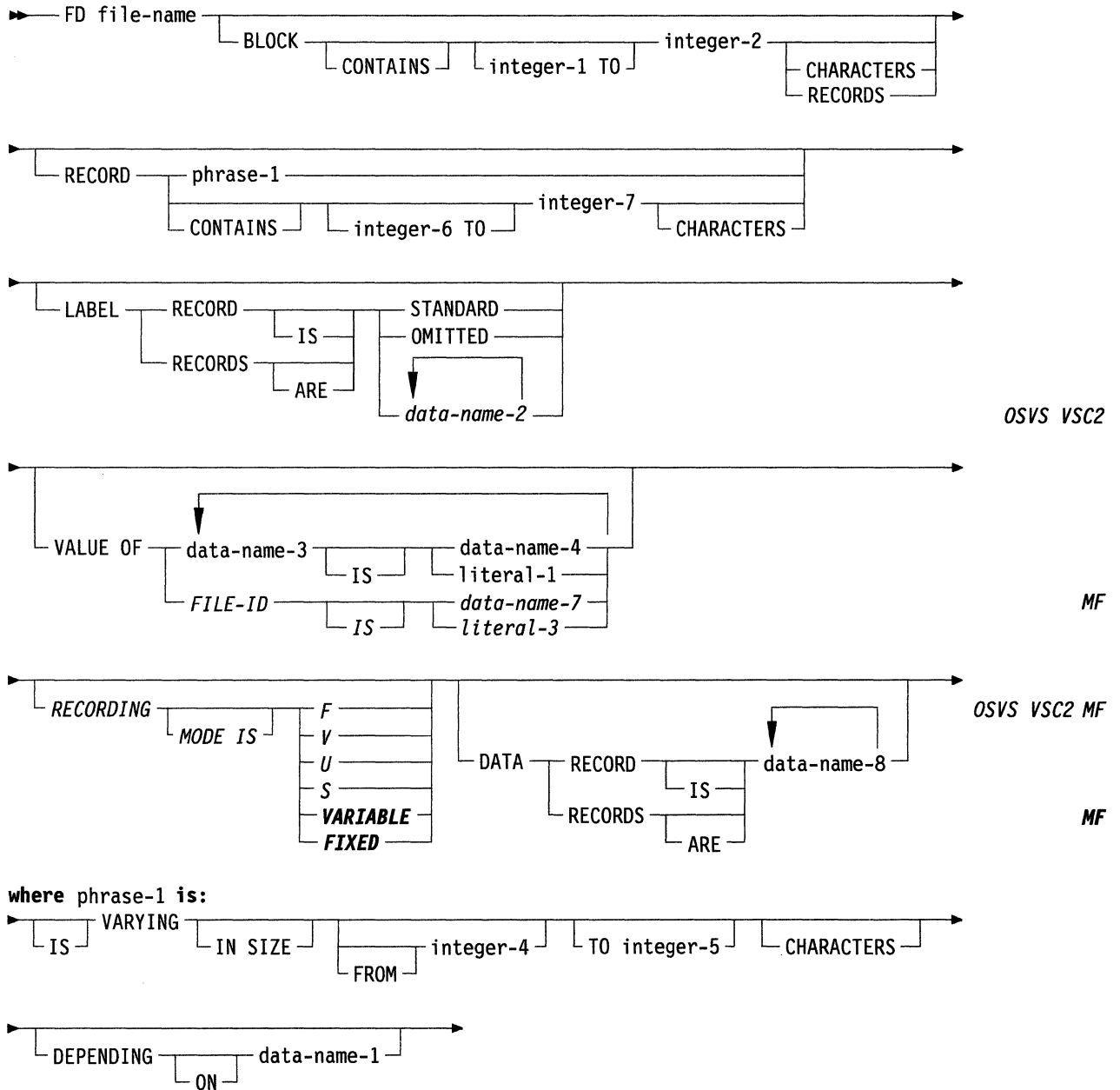
File Description — Complete Entry Skeleton

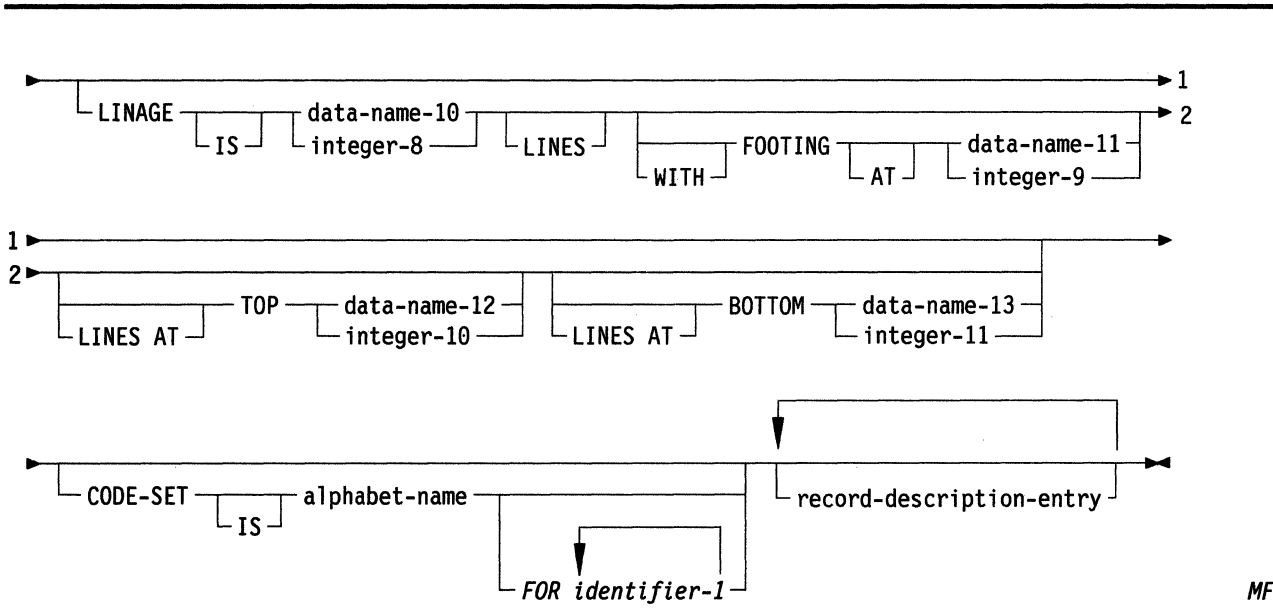
The file description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

General Formats

The following figures show the general formats of file description:

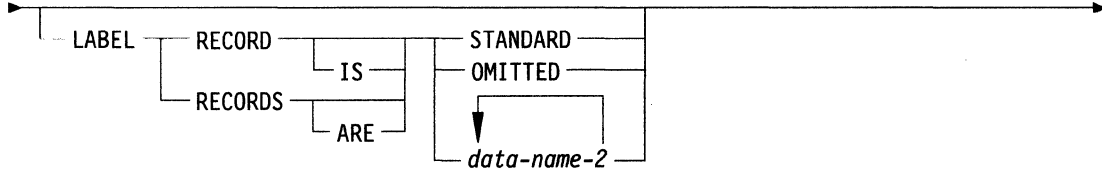
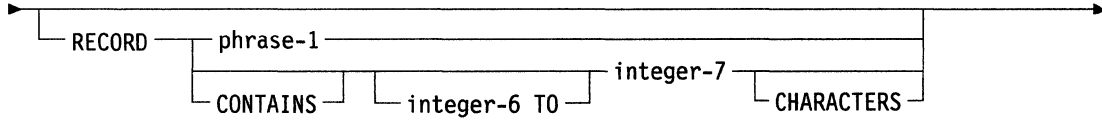
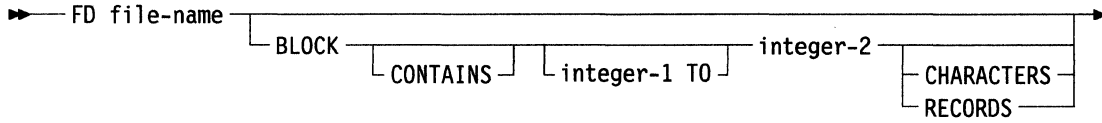
Format 1 (Sequential Files)



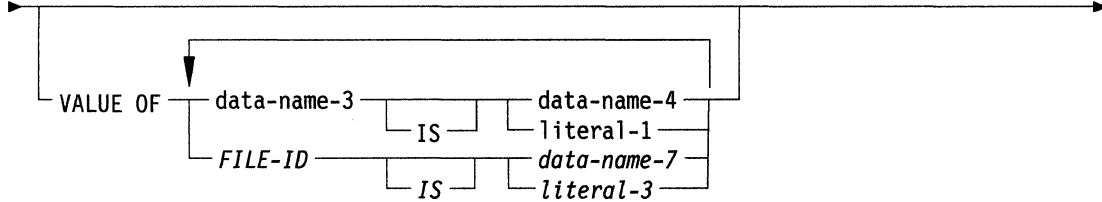


MF

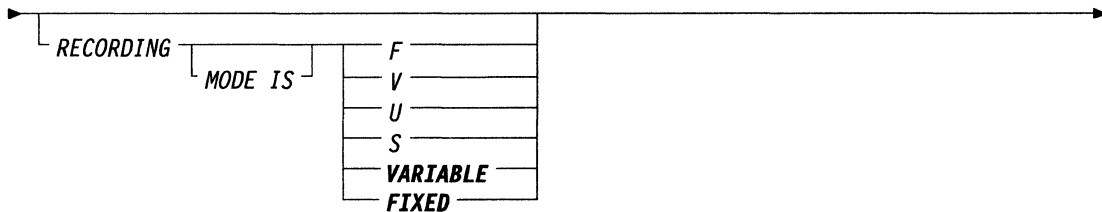
Format 2 (Relative and Indexed Files)



OSVS VSC2

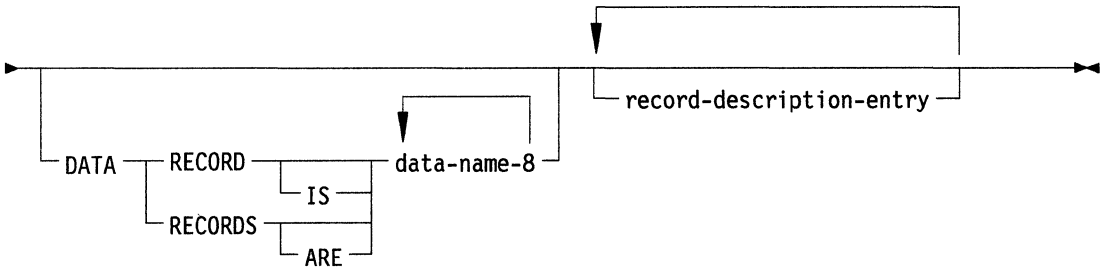


MF

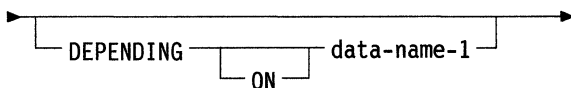
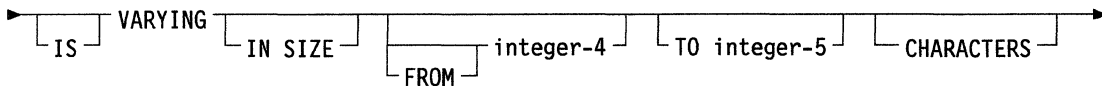


OSVS VSC2 MF

MF



where phrase-1 is:



Syntax Rules

The following syntax rules apply to file description:

All Formats (All Files)

1. The level indicator FD identifies the beginning of a file description and must precede the file name.
2. The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial. All clauses are optional.
3. One or more record description entries must follow the file description entry.
4. *If the VALUE OF FILE-ID clause is specified, literal-3 must be a nonnumeric literal and cannot be a figurative constant.* MF

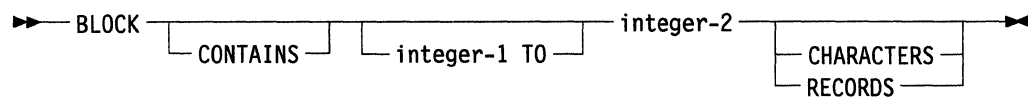
BLOCK CONTAINS Clause

Function

The BLOCK CONTAINS clause specifies the size of a physical record.

General Format

The following figure shows the general format of the BLOCK CONTAINS clause:



General Rule

This clause is required for documentation purposes only.

CODE-SET Clause

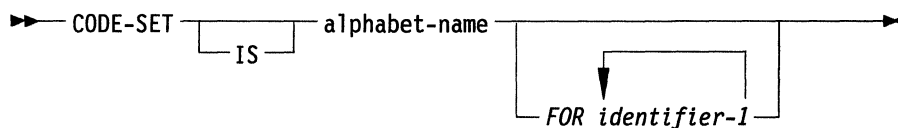
Function

The CODE-SET clause specifies the character code set used to represent data on the external media.

It may only be specified for files with sequential organization.

General Format

The following figure shows the general format of the CODE-SET clause:



MF

Syntax Rules

The following syntax rules apply to the CODE-SET clause:

1. When the CODE-SET clause is specified for a file, all data in the file must be described as USAGE IS DISPLAY and any signed numeric data must be described with the SIGN IS SEPARATE clause.
This restriction does not apply. MF
2. The alphabet-name clause referenced by the CODE-SET clause must not specify the literal phrase.
This restriction does not apply. MF
3. The CODE-SET clause may only be specified for nondisk files.
4. *identifier-1 can be qualified, but must not be subscripted.* MF
5. *Each identifier-1 must be a data item described in a record description for the file and must not be a record description. All identifier-1s must be described in the same record description.*
6. *If the optional FOR phrase is specified, the CODE-SET clause specifies the character code to be used for the data items named. If the FOR phrase is not specified, the CODE-SET clause specifies the character code to be used for the whole file.*
7. *The native character set is assumed to be used for any file, or data item in a file, to which no CODE-SET clause applies. The native character set is ASCII by default but can be changed by an AIX VS COBOL system directive as described in the User's Guide.*

8. *The data in the record area is always in ASCII. If alphabet-name-1 has been equated in the SPECIAL NAMES to EBCDIC, then data affected by the CODE-SET clause is translated from ASCII to EBCDIC as it is written to the file, or from EBCDIC to ASCII as it is read from the file. If alphabet-name-1 has been equated in the SPECIAL NAMES to STANDARD-1, STANDARD-2, NATIVE, or ASCII, no translation is necessary.*
9. *For the purposes of this translation, any data item to which a CODE-SET applies is treated as alphanumeric. No account is taken of the class and category of the item as described in its data description.*
10. *If identifier-1 has an OCCURS clause, the CODE-SET clause applies to only the first occurrence of it. If identifier-1 has a subordinate item with an OCCURS clause, the CODE-SET clause applies to the whole of identifier-1.*

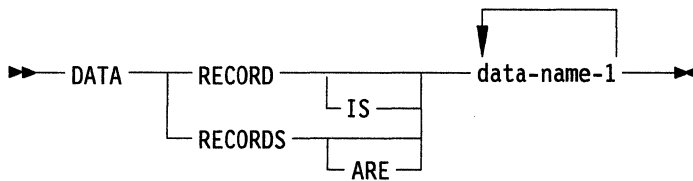
DATA RECORDS Clause

Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

General Format

The following figure shows the general format of the DATA RECORDS clause.



Syntax Rule

Each data-name-1 is the name of a data record and should have the 01 level number record description, with the same name, associated with it.

General Rules

The following general rules apply to the DATA RECORDS clause:

1. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.
2. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

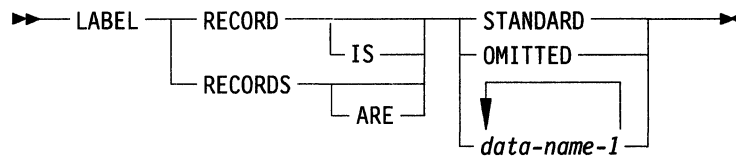
LABEL RECORDS Clause

Function

The LABEL RECORDS clause specifies whether labels are present.

General Format

The following figure shows the general format of the LABEL RECORDS clause:



OSVS VSC2

Syntax Rules

The following syntax rules apply to the LABEL RECORDS clause:

1. This clause is not required in every file description entry.
2. *data-name-1* is the name of a label record and should have the 01 level number record description with the same name associated with it. OSVS VSC2
3. *data-name-1* must not appear in the DATA RECORDS clause for the file. OSVS VSC2

General Rule

This clause is used for documentation purposes only.

LINAGE Clause

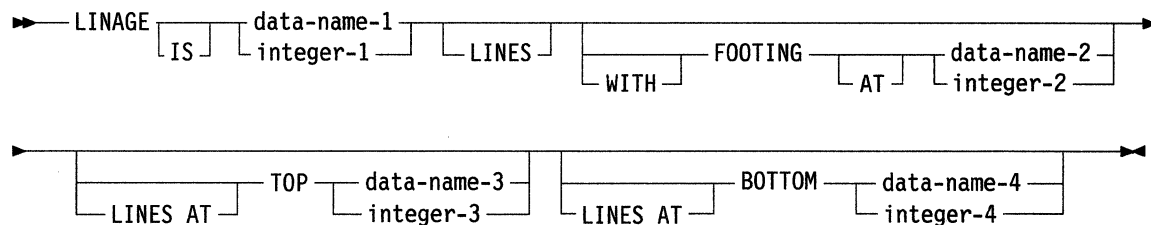
Function

The LINAGE clause provides a means for specifying the depth of a logical page in number of lines. It also provides a means for specifying the size of the top and bottom margins on the logical page and the line number within the page body at which the footing area begins.

The LINAGE clause may only be specified for files with sequential organization.

General Format

The following figure shows the general format for the LINAGE clause:



Syntax Rules

The following syntax rules apply to the LINAGE clause:

1. data-name-1, data-name-2, data-name-3, and data-name-4 must reference elementary unsigned numeric integer data items.
2. The value of integer-1 must be greater than zero.
3. The value of integer-2 must not be greater than integer-1.
4. The value of integer-3 and integer-4 may be zero.

General Rules

The following general rules apply to the LINAGE clause:

1. The LINAGE clause provides a means for specifying the size of a logical page in number of lines. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the values of these items are zero. If the FOOTING phrase is not specified, the assumed value is equal to integer-1, or the contents of the data item referenced by data-name-1, whichever is specified.

There is not necessarily any relationship between the size of the logical page and the size of a physical page.

-
2. The value of integer-1, or the data item referenced by data-name-1, specifies the number of lines that can be written and/or spaced on the logical page. The value must be greater than zero. The part of the logical page in which these lines can be written and/or spaced is called the page body.
 3. The value of integer-3, or the data item referenced by data-name-3, specifies the number of lines that comprise the top margin on the logical page. The value may be zero.
 4. The value of integer-4, or the data item referenced by data-name-4, specifies the number of lines that comprise the bottom margin on the logical page. The value may be zero.
 5. The value of integer-2, or the data item referenced by data-name-2, specifies the line number within the page body at which the footing area begins. The value must be greater than zero and not greater than the value of integer-1 or the data item referenced by data-name-1.

The footing area comprises the area of the logical page between the line represented by the value of integer-2 or the data item referenced by data-name-2, and the line represented by the value of integer-1 or the data item referenced by data-name-1, inclusive.

6. The value of integer-1, integer-3, and integer-4, if specified, will be used, at the time the file is opened by the execution of an OPEN statement with the OUTPUT phrase, to specify the number of lines that comprise each of the indicated sections of a logical page. The value of integer-2, if specified, will be used at that time to define the footing area. These values are used for all logical pages written for the file during a given execution of the program.
7. The values of the data items referenced by data-name-1, data-name-3, and data-name-4, if specified, will be used as follows:
 - a. The values of the data items, at the time an open statement with the OUTPUT phrase is executed for the file, will be used to specify the number of lines to comprise each of the indicated sections for the first logical page.
 - b. The values of the data items, at the time a WRITE statement with the ADVANCING PAGE phrase is executed or page overflow condition occurs (see "WRITE Statement" on page 8-89), will be used to specify the number of lines to comprise each of the indicated sections for the next logical page.
8. The value of the data item referenced by data-name-2, if specified, at the time an OPEN statement with the OUTPUT phrase is executed, will be used to define the footing area for the first logical page. At the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, it will be used to define the footing area for the next logical page.
9. A LINAGE-COUNTER is generated by the presence of a LINAGE clause. The value in the LINAGE-COUNTER at any given time represents the line number at which the device is positioned within the current page body. The rules governing the LINAGE-COUNTER are as follows:
 - a. A separate LINAGE-COUNTER is supplied for each file described in the FILE SECTION whose file description entry contains a LINAGE clause.
 - b. LINAGE-COUNTER may be referenced, but may not be modified, by Procedure Division statements. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file name when necessary.

-
- c. **LINAGE-COUNTER** is automatically modified, according to the following rules, during the execution of a **WRITE** statement to an associated file:
- When the **ADVANCING PAGE** phrase of the **WRITE** statement is specified, the **LINAGE-COUNTER** is automatically reset to one.
 - When the **ADVANCING identifier-2** or integer phrase of the **WRITE** statement is specified, the **LINAGE-COUNTER** is incremented by integer or the value of the data item referenced by identifier-2.
 - When the **ADVANCING** phrase of the **WRITE** statement is not specified, the **LINAGE-COUNTER** is incremented by the value one. Refer to “**WRITE Statement**” on page 8-89.
 - The value of **LINAGE-COUNTER** is automatically reset to one when the device is repositioned to the first line that can be written on for each of the succeeding logical pages. Refer to “**WRITE Statement**” on page 8-89.
- d. The value of **LINAGE-COUNTER** is automatically set to one at the time an **OPEN** statement is executed for the associated file.
10. Each logical page is contiguous to the next with no additional spacing provided.

RECORD Clause

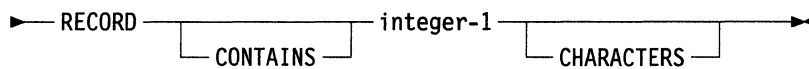
Function

The RECORD clause specifies either the number of character positions in a fixed-length record or the range of character positions in a variable-length record. If the number of character positions does vary, the clause specifies the minimum and maximum number of character positions.

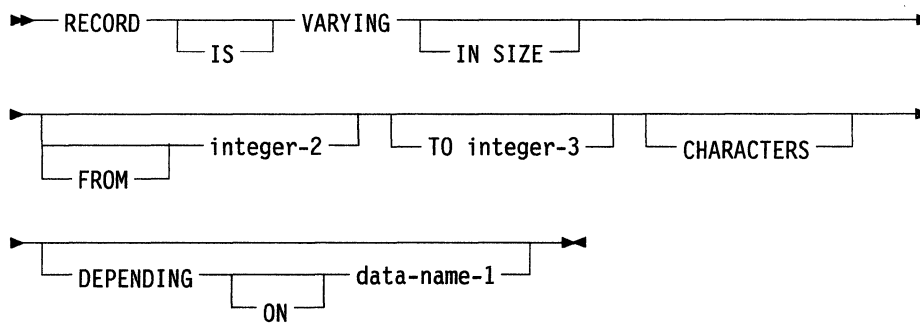
General Format

The following figures show the general format of the RECORD clause:

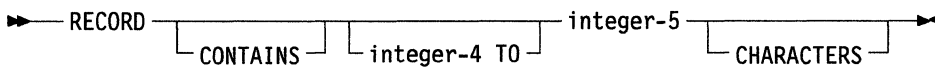
Format 1



Format 2



Format 3



Syntax Rules

The following syntax rules apply to the RECORD clause:

Format 1

1. No record description entry for the file may specify a number of character positions greater than integer-1.

Format 2

2. Record descriptions for the file must not describe records containing a lesser number of character positions than specified by integer-2 or records containing a greater number of character positions than specified by integer-3.
3. integer-3 must be greater than integer-2.
4. data-name-1 must describe an elementary unsigned integer in the WORKING-STORAGE or LINKAGE SECTION.

General Rules

The following general rules apply to the RECORD clause:

All Formats

1. If the RECORD clause is not specified, the size of each data record is completely determined by the record description entry.
2. If the associated file connector is an external file connector, all file description entries in the run-unit which are associated with that file connector must specify the same values for integer-1, integer-2, and integer-3. If the RECORD clause is not specified, all record description entries associated with this file connector must be the same length.

Format 1

3. Format 1 is used to specify fixed-length records. integer-1 specifies the number of character positions contained in each record in the file.

Format 2

4. Format 2 is used to specify variable-length records. integer-2 specifies the minimum number of character positions to be contained in any record of the file. integer-3 specifies the maximum number of character positions in any record of the file.
5. 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:
 - a. 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.
 - b. 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.
6. If integer-2 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.
7. If integer-3 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.
8. If data-name-1 is specified, 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.
9. If data-name-1 is specified, the execution of a DELETE, RELEASE, REWRITE, START, or WRITE statement or the unsuccessful execution of a READ or RETURN statement does not alter the content of the data item referenced by data-name-1.

-
10. During the execution of a RELEASE, REWRITE, or WRITE statement, the number of character positions in the record is determined by the following conditions:
 - a. If data-name-1 is specified, by the content of the data item referenced by data-name-1.
 - b. If data-name-1 is not specified and the record does not contain a variable occurrence data item, by the number of character positions in the record.
 - c. If data-name-1 is not specified and the record contains a variable occurrence data item, by the sum of the fixed portion and the portion of the table described by the number of occurrences at the time of execution of the output statement.
 11. If data-name-1 is specified, after the successful execution of a READ or RETURN statement, the contents of the data item referenced by data-name-1 indicate the number of character positions in the record just read.
 12. If the INTO phrase is specified in the READ or RETURN statement, the number of character positions in the current record treated as the sending data items in the implicit MOVE statement is determined by the following conditions:
 - a. If data-name-1 is specified, by the content of the data item referenced by data-name-1.
 - b. If data-name-1 is not specified, by the value that would have been moved into the data item referenced by data-name-1 had data-name-1 been specified.

Format 3

13. When Format 3 of the RECORD clause is used, integer-4 refers to the minimum number of characters in the smallest size data record and integer-5 refers to the maximum number of characters in the largest size data record. However, in this case, the size of each data record is completely defined in the record description entry.
14. The size of each data record is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the total number of characters in all fixed-length elementary items, plus the sum of the maximum number of characters in any variable-length item subordinate to the record. This sum may be different from the actual size of the record. Refer to "Selection of Character Representation and Radix" on page 2-20, "SYNCHRONIZED Clause" on page 6-37, and "USAGE Clause" on page 6-39.

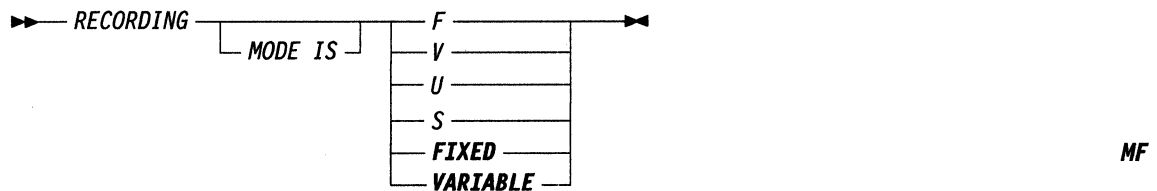
RECORDING MODE Clause

Function

The RECORDING MODE clause specifies the format of the logical records in the file. OSVS VSC2
MF

General Format

The following figure shows the general format of the RECORDING MODE clause. OSVS VSC2
MF



General Rules

The following general rules apply to the RECORDING MODE clause: OSVS VSC2

1. *Specifying RECORDING MODE IS F(ixed) causes all the records in the file to be the same length.*
2. *If RECORDING MODE IS V(ariable) is specified, the records in the file may be either fixed or variable in length. Each data record includes a record length field. These fields are not part of the record description.*
3. *The U (undefined) and S (spanned) options are used for documentation purposes only.*

VALUE OF Clause

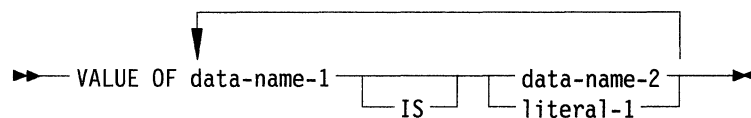
Function

The VALUE OF clause specifies the description of an item in the label records associated with a file.

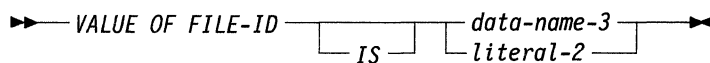
General Format

The following figures show the general format of the VALUE OF clause:

Format 1 (All Files)



Format 2 (All Files)



MF

Syntax Rules

The following syntax rules apply to the VALUE OF clause:

Format 1 (All Files)

1. data-name-2 should be qualified when necessary but cannot be subscripted or indexed, nor can it be an item described with the USAGE IS INDEX clause.
2. data-name-2 must be in the WORKING-STORAGE SECTION.

Format 2 (All Files)

3. *literal-2 must be a nonnumeric literal and cannot be a figurative constant.* MF
4. *The VALUE OF FILE-ID clause cannot be used if external file reference, data-name-3, or literal-2 has been specified in the ASSIGN clause in the FILE-CONTROL entry. Refer to "FILE-CONTROL Entry" on page 8-18.* MF

General Rules

The following general rules apply to the VALUE OF clause:

Format 1 (All Files)

1. This clause is used for documentation purposes only.
2. A figurative constant may be substituted in the format above wherever a literal is specified.

Format 2 (All Files)

3. *The character-string specified in literal-2 or data-name-3 is taken as the external file name.* *MF*

Procedure Division for File Input and Output

CLOSE Statement

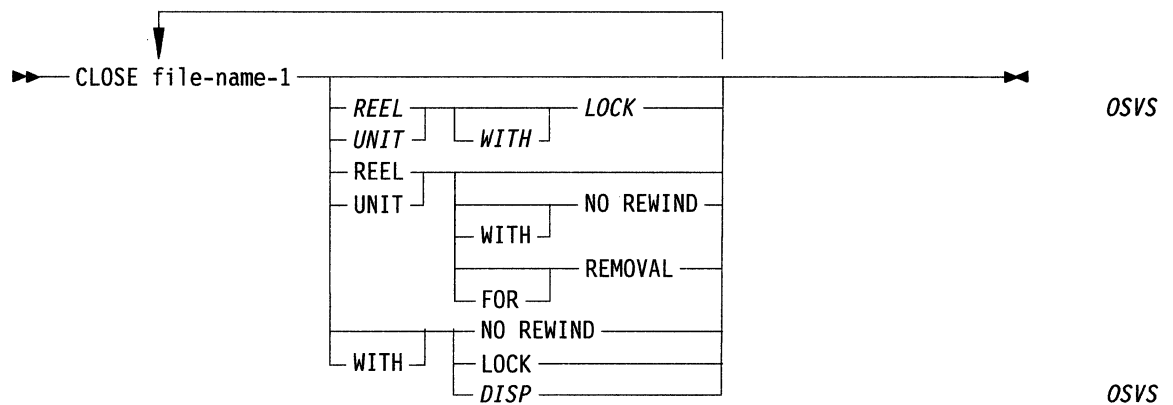
Function

The CLOSE statement terminates the processing of reels or units and files, with optional rewind and/or lock or lock removal where applicable.

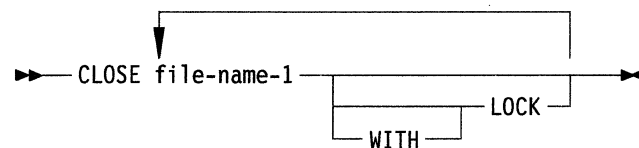
General Format

The following figures show the general format of the CLOSE statement:

Format 1 (Sequential Files)



Format 2 (Relative And Indexed Files)



Syntax Rules

The following syntax rules apply to the CLOSE statement:

All Formats (All Files)

1. The files referenced in the CLOSE statement need not all have the same organization or access.

Format 1 (Sequential Files)

2. *The REEL and UNIT phrases should be specified only for files with MULTIPLE REEL or MULTIPLE UNIT specified in their SELECT clause, except when used with the FOR REMOVAL option. The statements CLOSE REEL and CLOSE UNIT are null statements. They are for documentary purposes only. It is important that no other files are open on the device(s) closed by a CLOSE REEL or CLOSE UNIT statement.* MF
- The statements CLOSE REEL WITH LOCK and CLOSE UNIT WITH LOCK are treated as equivalent to CLOSE REEL FOR REMOVAL.* OSVS
3. *The DISP option is only applicable to tape files. It is for documentation only.* OSVS

General Rules

The following general rules apply to the CLOSE statement:

All Formats (All Files)

1. A CLOSE statement may only be executed for a file in an open mode.
2. Following the successful execution of a CLOSE statement, the record area associated with file name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.
3. The action taken if a file is in the open mode when a STOP RUN statement is executed is to close the file. The action taken for a file opened in a called program and not closed prior to the execution of a CANCEL statement for the program is to close the file.
4. *Following the successful execution of a CLOSE statement, all record or file locks held by the run-units on the closed file are released.* MF
5. The execution of the CLOSE statement causes the value of the I-O status associated with file-name-1 to be updated. Refer to "I-O Status" on page 8-8.
6. The results of executing each type of CLOSE statement for each category of file are summarized in Figure 8-2 on page 8-57.

The following definitions explain the symbols you will find in the table. Where the definition depends on whether the file is an input, output, or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

A Previous Reels or Units Unaffected

Input Files and Input-Output Files:

All reels or units in the file prior to the current reel or unit are processed except those reels or units controlled by a prior CLOSE REEL or UNIT statement. If the current reel or unit is not the last in the file, the reels or units in the file following the current one are not processed.

Output Files:

All reels or units in the file prior to the current reel or unit are processed except those reels or units controlled by a prior CLOSE REEL or UNIT statement.

B No Rewind of Current Reel

The current reel or unit is left in its current position.

C Close File

Input Files and Input-Output Files (Sequential Access Mode):

If the file is positioned at its end and label records are specified for the file, the labels are processed according to the operating system label convention. The result of the CLOSE statement is undefined when label records are specified but not present, or when label records are not specified but are present. If the file is positioned at its end and label records are not specified for the file, label processing does not take place, but other closing operations dependent on the Run Time Environment are executed. Refer to the *User's Guide*. If the file is positioned other than at its end, the closing operations dependent on the RTE are executed but there is no end label processing.

Input Files and Input-Output Files (Random or Dynamic Access Mode); Output Files (Random, Dynamic or Sequential Access Mode):

If label records are specified for the file, the labels are processed according to the operating system standard label convention. The result of the CLOSE statement is undefined when label records are specified but not present, or when label records are not specified but are present. If label records are not specified for the file, label processing does not take place but other closing operations dependent on the RTE are executed.

D Reel or Unit Removal

The reel or unit may be accessed again, in its proper order of reels or units within the file, if a CLOSE statement without the REEL or UNIT phrase is executed for this file followed by an OPEN statement for the file.

E File Lock

This file cannot be opened again during this execution of this run-unit.

F Close Reel or Unit

Input Files and Input-Output Files:

The following operations take place:

- a. If the current reel or unit is the last or only reel or unit for the file, or the reel is on a nonreel or unit medium, there is no reel or unit swap.
- b. If another reel or unit exists for the file, a reel or unit swap occurs and the standard beginning reel or unit label procedure is executed. If no data records exist for the current volume, another reel or unit swap occurs.

Output Files (Reel or Unit Media):

The following operations take place:

- a. The standard ending reel or unit label procedure is executed.
- b. A reel or unit swap.
- c. The standard beginning reel or unit label procedure is executed.

-
- d. The next executed WRITE statement that references the file directs the next logical data record to the next reel or unit of the file.

Output Files (Nonreel or Unit Media):

Execution of this statement is considered successful. The file remains in the open mode, and no action takes place except as specified in general rule 5.

G Rewind

The current reel or analogous device is positioned at its physical beginning.

H Optional Phrases Ignored

The CLOSE statement is executed as if none of the optional phrases is present.

X Illegal

This is an illegal combination of a CLOSE option and a file category. The results at object time are undefined.

Format 1 (Sequential Files)

Except where otherwise stated in the general rules below, the terms reel and unit are synonymous and completely interchangeable in the CLOSE statement. Treatment of sequential mass storage files is logically equivalent to the treatment of a file on tape or analogous sequential media.

7. For the purpose of showing the effect of various types of CLOSE statements as applied to various storage media, all files are divided into the following categories:
 - a. Nonreel or unit. A file with input or output medium such that the concepts of rewind and reels or units have no meaning.
 - b. Sequential single reel or unit. A sequential file entirely contained on one reel or unit.
 - c. Sequential multiple reels or units. A sequential file contained on more than one reel or unit.
8. If the OPTIONAL phrase was specified in the FILE-CONTROL paragraph of the Environment Division and the file is not present, standard end-of-file processing is not performed for that file.
9. When a CLOSE statement without the REEL or UNIT phrase is executed for a file, no other statement can be executed referencing that file, explicitly or implicitly, unless an intervening OPEN statement for the file is executed. (The SORT or MERGE statements with the USING or GIVING phrases are exceptions.)
10. The WITH NO REWIND and FOR REMOVAL phrases have no effect at object time if they do not apply to the storage media on which the file resides.
11. If WITH LOCK is specified, the file cannot be reopened in the current execution of the run-unit, provided that your run time environment supports this option. Otherwise a normal CLOSE takes effect. (*This option has no connection with the record or file locking used when sharing files.*) *MF*

Format 2 (Relative And Indexed Files)

12. Relative and indexed files are classified as belonging to the category of nonsequential single or multiple reels or units.
13. If a CLOSE statement was executed for a file, no other statement can be executed referencing that file, either explicitly or implicitly, unless an intervening OPEN statement is executed.
14. If WITH LOCK is specified, the file cannot be reopened in the current execution of the run-unit.

CLOSE Statement Format	File Category			
	Nonreel or Unit	Sequential Single Reel or Unit	Sequential Multiple Reel or Unit	Non-Sequential Single or Multiple Reels or Units
CLOSE	C	C, G	A, C, G	C
CLOSE WITH LOCK	C, E	C, E, G	A, C, E, G	C, E
CLOSE WITH NO REWIND	C, H	B, C	A, B, C	X
CLOSE REEL OR UNIT	F	F, G	F, G	X
CLOSE REEL OR UNIT FOR REMOVAL	F	D, F, G	D, F, G	X
CLOSE REEL OR UNIT WITH NO REWIND	X	X	F, B	X

Figure 8-2. Relationship of CLOSE Statement with File Category

COMMIT Statement

Function

The COMMIT statement releases record locks.

MF

General Format

The following figure shows the general format of the COMMIT statement:

MF

►— COMMIT —►

General Rules

The following general rules apply to the COMMIT statement:

MF

- 1. Execution of the COMMIT statement causes all record locks in all files held by the run-unit to be released.*
- 2. The file lock on an exclusive file is not affected by the COMMIT statement.*

DELETE Statement

Function

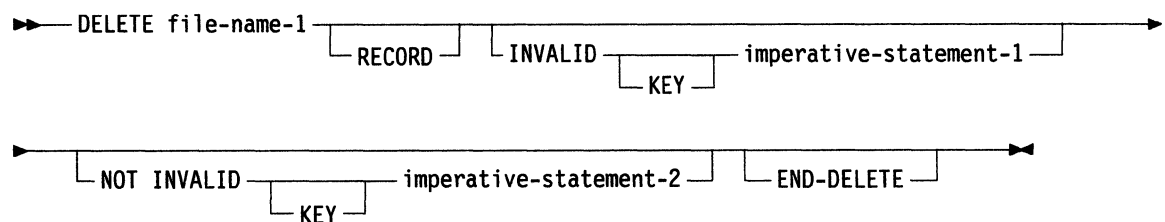
The DELETE statement logically removes a record from a mass storage file. It may only be specified for files with relative or indexed organization.

The DELETE FILE statement physically removes the specified files from the physical devices on which they reside. MF

General Format

The following figures show the general format of the DELETE statement:

Format 1



Format 2



MF

Syntax Rules

The following syntax rules apply to the DELETE statement:

1. The INVALID KEY phrase must not be specified for a DELETE statement referencing a file in sequential access mode.
2. The INVALID KEY phrase must be specified for a DELETE statement referencing a file which is not in sequential access mode and for which an applicable USE AFTER STANDARD EXCEPTION PROCEDURE is not specified.

General Rules

The following general rules apply to the DELETE statement:

Format 1

1. The associated file must be open in the I-O mode at the time this statement is executed. Refer to "OPEN Statement" on page 8-62.
2. For files in sequential access mode, the last input-output statement executed for the file name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The operating system logically removes the record accessed by that READ statement from the file.
3. For a file in random or dynamic access mode, the operating system logically removes the record identified by the contents of a KEY data item associated with the file name from the file. For a relative file, this KEY data item is the RELATIVE KEY, and for an indexed file, it is the prime RECORD KEY. If the file does not contain the record specified by the key, an INVALID key condition exists. Refer to "INVALID KEY Condition" on page 8-11.
4. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.
5. The execution of a DELETE statement does not affect the contents of the record area associated with the file name.
6. The file position indicator is not affected by the execution of a DELETE statement.
7. The execution of the DELETE statement causes the value of the specified FILE STATUS data item associated with the file name to be updated. Refer to "I-O Status" on page 8-8.
8. Transfer of control following the successful or unsuccessful execution of the DELETE operation depends on the presence or absence of the INVALID KEY phrase and NOT INVALID KEY phrase in the DELETE statement. Refer to "INVALID KEY Condition" on page 8-11.
9. The END-DELETE phrase delimits the scope of the DELETE statement. Refer to "Explicit and Implicit Scope Terminators" on page 2-36.
10. *When using DELETE, the record to be deleted must not be locked by another run-unit.* MF
11. *Following the successful execution of a DELETE statement, any record lock held by the run-unit on the deleted record is released.* MF
12. The execution of the DELETE statement causes the value of the I-O status associated with file name 1 to be updated. Refer to "I-O Status" on page 8-8.

Format 2

13. *The specified files must be closed when the statement is executed.* MF

Example

The following example shows the DELETE statement:

```
OPEN I-O ACCT-PAYABLE.  
  :  
  :  
READ ACCT-PAYABLE RECORD INTO ACCT-PAY-RECORD.  
IF ACCT-PAY-STATUS OF ACCT-PAY-RECORD = "PAID"  
  DELETE ACCT-PAYABLE RECORD.
```

To delete a record, the associated file must be opened in I-O mode, and a successful READ must precede the DELETE statement.

OPEN Statement

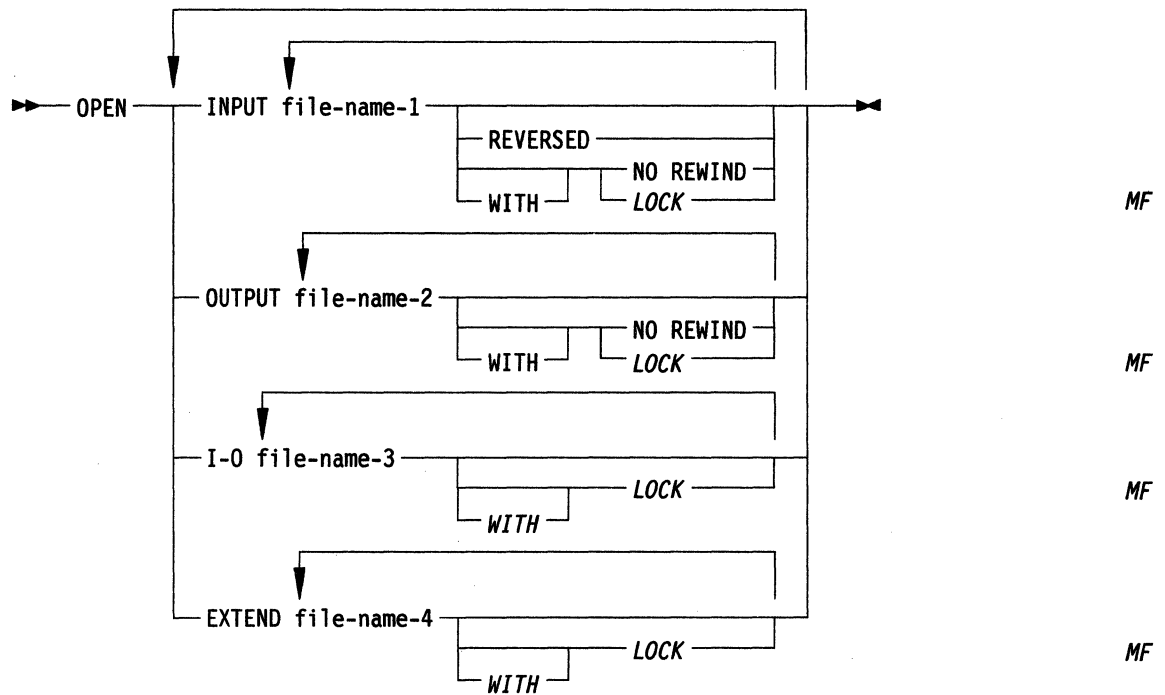
Function

The OPEN statement initiates the processing of files. It also performs checking and/or writing of labels and other input-output operations.

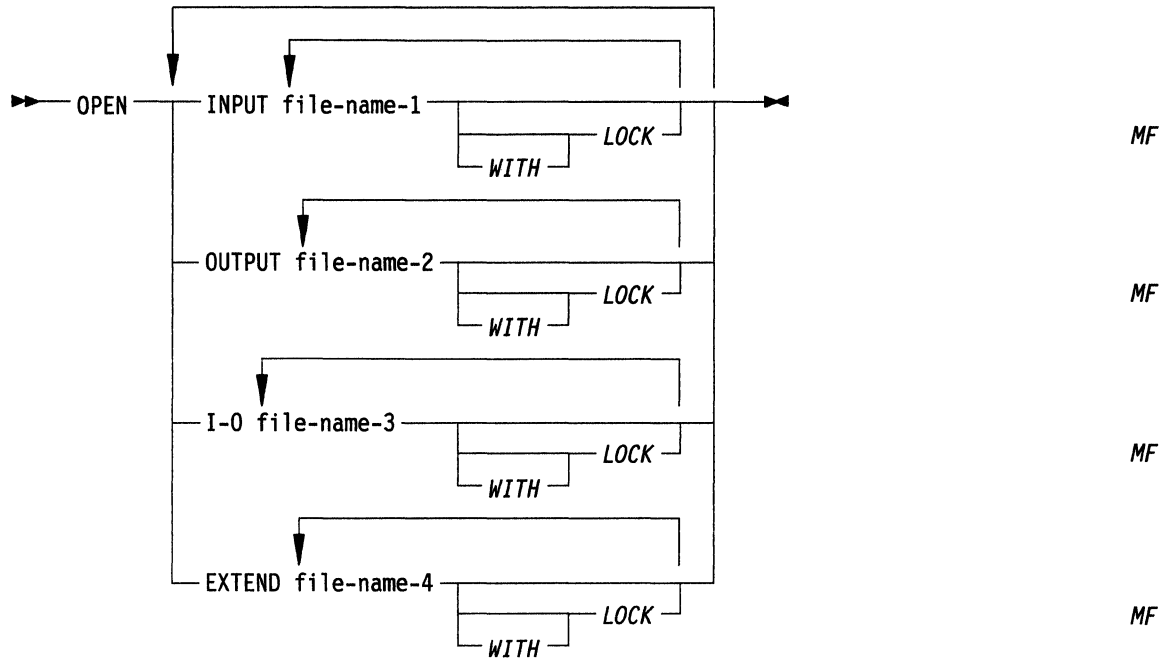
General Format

The following figures show the general format of the OPEN statement:

Format 1 (Sequential Files)



Format 2 (Relative and Indexed Files)



Syntax Rules

The following syntax rules apply to the OPEN statement:

All Formats (All Files)

1. The files referenced in the OPEN statement need not all have the same organization or access.

Format 1 (Sequential Files)

2. The REVERSED and NO REWIND phrases can only be used with record-sequential files. NO REWIND is used for documentation purposes only.
3. The I-O phrase can be used only for disk files. The I-O phrase cannot be used for files in line-sequential organization.
4. The EXTEND phrase must be used only for files for which the LINAGE clause has not been specified.
5. The EXTEND phrase must not be specified with multiple file reels.

Format 2 (Relative and Indexed Files)

6. The EXTEND phrase must only be used for files in the sequential access mode.

General Rules

The following general rules apply to the OPEN statement:

All Formats (All Files)

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.
2. The successful execution of an OPEN statement makes the associated record area available to the program.
3. Prior to the successful execution of an OPEN statement for a given file, no statement (except for a SORT or MERGE statement with the USING or GIVING phrases) can be executed that references that file, either explicitly or implicitly.
4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input/output statements. In the table on page 8-66, X at an intersection indicates that the specified statement used in the access mode given for that row may be used with the file organization and open mode given at the top of the column.
5. *If the WITH LOCK phrase is specified, the OPEN statement acquires a lock on the whole file. (This is equivalent to specifying LOCK MODE IS EXCLUSIVE in the SELECT statement for the file.)* *MF*
6. A file may be opened with the INPUT, OUTPUT, EXTEND, and I/O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for the same file must be preceded by a CLOSE statement for that file.
7. Execution of the OPEN statement does not obtain or release the first data record.
8. The ASSIGNED name in the SELECT statement for a file is processed as follows:
 - a. When the INPUT phrase is specified, the execution of the OPEN statement causes the ASSIGNED name to be checked in accordance with the operating system conventions for opening files for input.
 - b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the ASSIGNED name to be written in accordance with the operating system conventions for opening files for output.
9. The file description entry for file-name-1, file-name-2, file-name-3, and file-name-4 must be equivalent to that used when the file was created.
10. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the file position indicator to the first record currently existing within the file. If no records exist in the file, the file position indicator will indicate the AT END condition for the next executed READ statement. If the file does not exist, OPEN INPUT causes an error status.
11. When the I-O phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:
 - a. The labels are checked for accordance with the operating system specified conventions for input/output label checking.
 - b. The new labels are written in accordance with the operating system specified conventions for input/output label writing.
12. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time, the associated file contains no data records. If a file of the same name exists, it is deleted. If it is write protected, an error occurs.

-
13. When the EXTEND phrase is specified, the OPEN statement positions the file pointer immediately after the last logical record for the file. The last logical record for a sequential file is the last record written in the file. The last logical record in a relative file is the currently existing record with the highest relative record number. The last logical record in an indexed file is the currently existing record with the highest prime key value.
 14. When the EXTEND phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:
 - a. The beginning file labels are processed only in the case of a single reel or unit file.
 - b. The beginning reel or unit labels on the last existing reel or unit are processed as though the file was being opened with the INPUT phrase.
 - c. The existing ending file labels are processed as though the file is being opened with the INPUT phrase. These labels are then deleted.
 - d. Processing then proceeds as though the file had been opened with the OUTPUT phrase.
 15. The execution of the OPEN statement causes the value of the FILE STATUS data item to be updated. Refer to "I-O Status" on page 8-8.
 16. *When LOCK MODE IS EXCLUSIVE is specified, successful execution of an OPEN statement locks the file exclusively to that run-unit.* MF
 17. *When LOCK MODE IS AUTOMATIC or LOCK MODE IS MANUAL is specified, the file referred to is shareable. More than one run-unit may successfully open such a file.* MF
 18. *A file opened for OUTPUT is implicitly defined as a file with an exclusive lock, meaning it is not shareable.* MF
 19. *Only shareable files opened for I-O can acquire record locks.* MF

Format 1 (Sequential Files)

20. If an input file is designated with the OPTIONAL phrase in its SELECT clause, the object program causes an interrogation for the presence or absence of this file. If the file is not present, the first READ statement for this file causes the AT END condition to occur.
21. If the storage medium for the file permits rewinding, the following rules apply:
 - a. Execution of the OPEN statement causes the file to be positioned at its beginning.
 - b. When the REVERSED phrase is specified, the file is positioned at its end by execution of the OPEN statement.
22. When the REVERSED phrase is specified, the following READ statements for the file make the data available in reversed order. In this case, the data would be read from the last record first.
23. When the EXTEND phrase is specified, the OPEN statement positions the file pointer immediately following the last logical record of that file. Subsequent WRITE statements referencing the file add records to the file as though the file had been opened with the OUTPUT phrase. If the file does not exist it is created.

24. When the EXTEND phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:
- The beginning file labels are processed only in the case of a single reel or unit file.
 - The beginning reel or unit labels on the last existing reel or unit are processed as though the file is opened with the INPUT phrase.
 - The existing ending file labels are processed as though the file is being opened with the INPUT phrase. These labels are then deleted.
 - Processing then proceeds as though the file had been opened with the OUTPUT phrase.
25. The I-O phrase permits the opening of a file for both input and output operations, *except for files with ORGANIZATION LINE-SEQUENTIAL*. If the file does not exist it is created and used as an empty file for input unless NOT OPTIONAL was specified in the SELECT statement. An attempt to WRITE it causes an error. MF

The following table shows examples of permissible combinations of statements and OPEN modes:

File Access Mode	Statement	Open Modes			
		Input	Output	I-O	Extend (Sequential Files)
Sequential	READ	X		X	X
	WRITE		X		
	REWRITE			X	
	START	X		X	
	DELETE			X	
Random (Nonsequential Files)	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
Dynamic (Nonsequential Files)	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

Notes:

- The START and DELETE statements may not be used for files with sequential organization.
- I-O mode is not supported for line-sequential files.

Example

The following is an example of OPEN statements in the Procedure Division.

```
OPEN INPUT PAYROLL-FILE, BENEFIT-FILE,  
      OUTPUT REPORT-FILE.  
OPEN I-O INVENTORY-FILE, SUPPLIER-FILE,  
      EXTEND TRANS-HISTORY.
```

READ Statement

Function

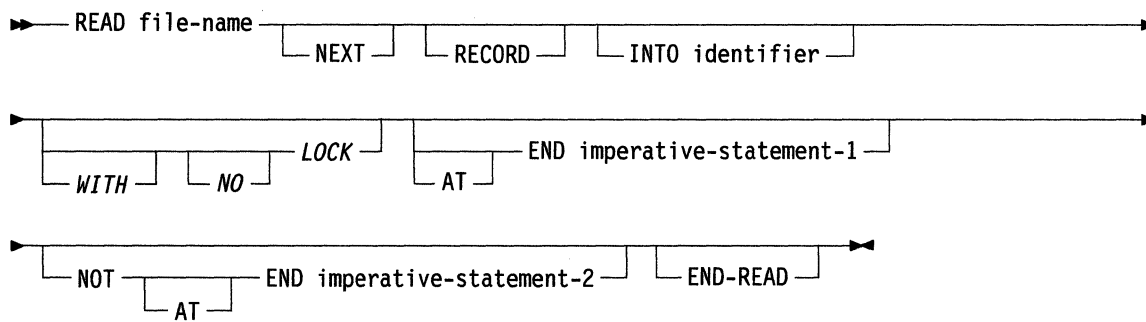
For sequential access, the READ statement makes available the next or previous logical record from a file. For random access, the READ statement makes available a specified record from a mass storage file.

MF

General Format

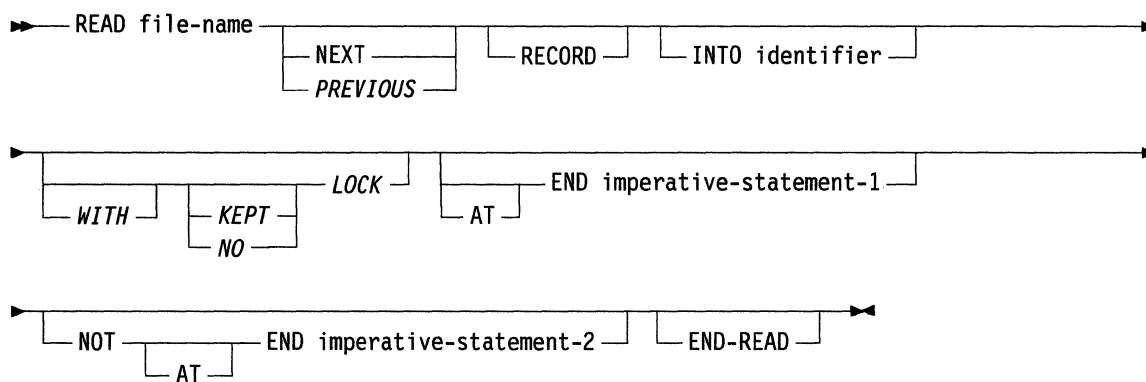
The following figures show the general format for the READ statement:

Format 1 (Sequential Files)



MF

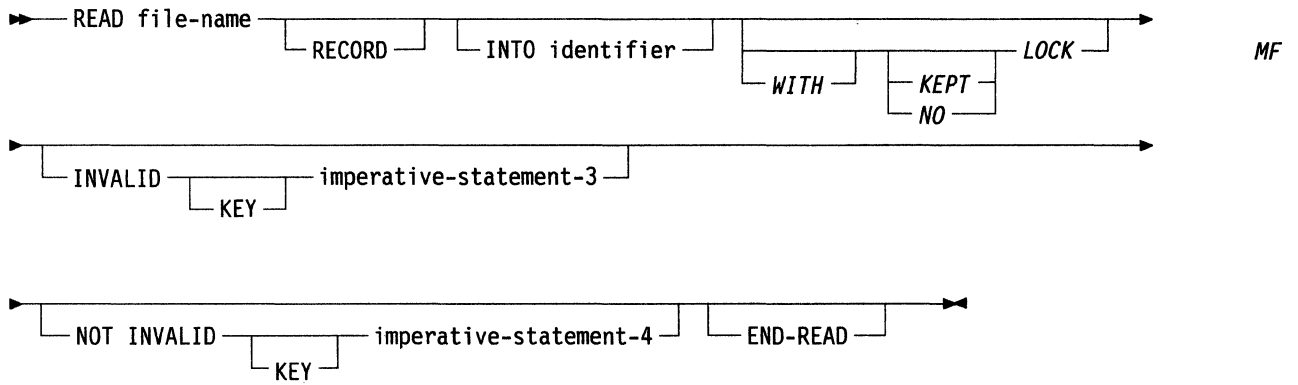
Format 2 (Relative and Indexed Files)



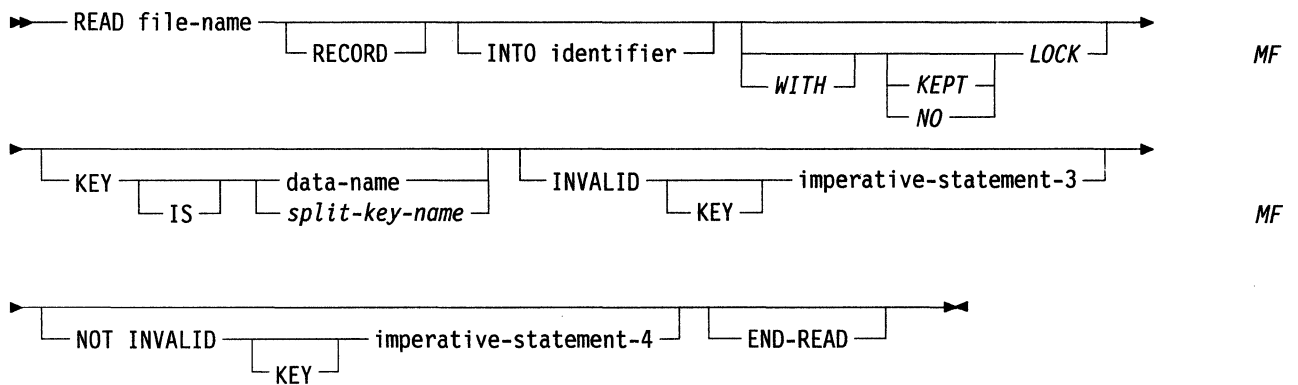
MF

MF

Format 3 (Relative Files)



Format 4 (Indexed Files)



Syntax Rules

The following syntax rules apply to the READ statement:

All Formats (All Files)

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The memory area associated with identifier and the memory area which is the record area associated with file name must not be the same memory area.
2. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file name.
3. *The WITH LOCK phrase must be included when single records are being locked manually in a shareable file.* MF
4. *The NO LOCK phrase is only allowed when records are being locked manually or automatically in a shareable file.* MF

DBCS Support

5. With the READ INTO phrase, the identifier (record area) may be a USAGE DISPLAY-1 item.

End of DBCS Support

Formats 1 and 2 (All Files)

- 6. The **NEXT** phrase can be specified for sequential files or files in sequential access mode. *The PREVIOUS phrase cannot be specified for sequential files.* **MF**
- 7. The **NEXT** or **PREVIOUS** phrases must be specified for files in dynamic access mode when records are to be retrieved sequentially. **MF**

Formats 3 and 4 (Relative And Indexed Files)

- 8. Format 3 or 4 is used for files in random access mode or in dynamic access mode when records are to be retrieved randomly.

Format 4 (Indexed Files)

- 9. Data-name must be the name of a data item specified as a record key associated with file name.
- 10. Data-name may be qualified.
- 11. *split-key-name-1 is a concatenation of one or more data items specified as a record key associated with file name.* **MF**

DBCS Support

- 12. Data-name (the **KEY IS** clause) may be defined as a **USAGE DISPLAY-1** (DBCS) item.
- 13. When the **RECORD KEY** clause specifies a DBCS item, a **KEY** specified on the **READ** statement must be a DBCS item.

End of DBCS Support

Formats 2, 3, and 4 (Relative And Indexed Files)

- 14. *The WITH KEPT LOCK phrase must be included when multiple records are being locked manually in a shareable file.* **MF**

General Rules

The following general rules apply to the READ statement:

All Formats (All Files)

1. The associated file must be open in the INPUT or I-O mode at the time this statement is executed. Refer to "OPEN Statement" on page 8-62.
2. For sequential files or files in sequential access mode, the NEXT phrase is optional and has no effect on the execution of the READ statement.
3. The execution of the READ statement causes the value of the FILE STATUS data item, associated with the file name to be updated. Refer to "I-O Status" on page 8-8.
4. Regardless of the method used to overlap access time with processing time, the concept of the READ statement continues to make a record available to the object program prior to the execution of any statement following the READ statement.
5. When the logical records of a file are described with more than one record description, these records automatically share the same memory area. This is equivalent to an implicit redefinition of the area. The contents of any data items lying beyond the range of the current data record are undefined at the completion of the READ statement execution.
6. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifiers is evaluated after the record has been read and immediately before it is moved to the data item.
7. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.
8. When the AT END condition is recognized, the following actions are taken in the specified order:
 - a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition. Refer to "I-O Status" on page 8-8.
 - b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative statement. Any USE procedure specified for this file is not executed.
 - c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file. That procedure is then executed.

When the AT END condition occurs, execution of the input/output statement which caused the condition is unsuccessful.

9. If an AT END condition does not occur during the execution of a READ statement and if a NOT AT END phrase is specified, control is transferred to imperative-statement-2 at the following appropriate time:
 - a. After the record is made available, after setting the file position indicator, after updating the value of the I-O status associated with file-name-1, and after executing any implicit move resulting from the presence of an INTO phrase.
 - b. If the record is not made available for a reason other than an AT END condition, after updating the value of the I-O status associated with file name, and after executing the procedure, if any, specified by a USE AFTER STANDARD EXCEPTION PROCEDURE statement applicable to file-name-1.
10. Following the unsuccessful execution of any READ statement, the content of the associated record area is undefined and the file position indicator is set to indicate that no next record has been established.

-
11. Execution of a *READ NEXT* or *READ PREVIOUS* statement does not update the file position indicator if a locked record is found. A subsequent *READ* statement will attempt to read the same record. See the *User's Guide* for compiler options and switches to override this behavior. MF

Format 1 (Sequential Files)

12. The record to be made available by the *READ* statement is determined as follows:
- a. If the file position indicator was positioned by the execution of the *OPEN* statement, the record pointed to is made available *unless the PREVIOUS option is specified*. In this case the *AT END* condition occurs as in general rule 20. MF
 - b. If the file position indicator was positioned by the execution of a previous *READ* statement, the file position indicator points to the next or, if the *PREVIOUS* option was specified, the previous existing record in the file. That record is made available. MF
13. If the end of a reel or unit is recognized during the execution of a *READ* statement, and end-of-file has not been reached, the following procedures are executed:
- a. The standard ending reel or unit label procedure
 - b. A reel or unit swap
 - c. The standard beginning reel or unit label procedure
 - d. The first data record of the new reel or unit is made available.
14. If a file described with the *OPTIONAL* clause is not present at the time the file is opened, then at the time of the execution of the first *READ* statement for the file, the *AT END* condition occurs and the execution of the *READ* statement is unsuccessful. The standard end-of-file procedures are not performed. Refer to "FILE-CONTROL Paragraph" on page 8-17, "OPEN Statement" on page 8-62, and "USE Statement" on page 8-86. Execution of the program then proceeds as in general rule 8.
15. For sequential files opened for *INPUT*, *READ*, or *READ WITH LOCK* statements do not acquire a record lock. MF
16. Two or more run-units can share a sequential output file by opening it *EXTEND* with *AUTOMATIC* or *MANUAL* record locking. Records appended to the file are in unspecified order. MF
17. For files opened for *I-O*:
- a. With *LOCK MODE AUTOMATIC*, unless the *WITH NO LOCK* phrase is specified, each record is locked as it is read and released again when the run-unit next accesses the file.
 - b. With *LOCK MODE MANUAL*, a simple *READ* statement does not acquire a lock on the record. The *READ WITH LOCK* statement must be used to acquire a record lock. The *WITH NO LOCK* phrase, if specified, is documentary.
18. If an end-of-file status occurs on a *READ* statement in a file opened for *I-O* or *INPUT* by one run-unit and opened *EXTEND* by another run-unit, the run-unit that attempted the *READ* must close the file. This run-unit has no access to the appended records because the status remains end-of-file.

Formats 1 and 2 (All Files)

19. If a *READ* statement with the *NEXT* option is executed, and no next logical record exists in the file, then the *AT END* condition occurs and the execution of the *READ* statement is considered unsuccessful. Refer to "I-O Status" on page 8-8.
20. If a *READ* statement with the *PREVIOUS* option is executed and no previous logical record exists in the file, the *AT END* condition occurs, and the execution of the *READ* statement is considered unsuccessful. MF

-
21. When the AT END condition has been recognized, *the next Format 1 or 2 READ statement executed for that file must be one of the following:* **MF**
- a. *A READ NEXT statement, if AT END occurred because no previous logical record existed*
 - b. *A READ PREVIOUS statement, if AT END occurred because no next logical record existed.*
- Otherwise, the AT END condition must be preceded by:*
- a. *A successful CLOSE statement followed by the execution of a successful OPEN statement for that file*
 - b. *A successful START statement for that file*
 - c. *A successful Format 3 (or Format 4 for indexed files) READ statement for that file.*
22. For a file for which dynamic access mode is specified, a READ statement with the NEXT phrase specified causes the next logical record to be retrieved from the file.
23. *For a file for which dynamic access mode is specified, a READ statement with the PREVIOUS phrase specified causes the previous logical record to be retrieved from the file as described in rule 12 on page 8-72 for sequential files or rule 25 for all other file types.* **MF**
24. If, at the time of execution of a READ statement, the position of file position indicator for that file is undefined, the execution of that READ is unsuccessful.

Formats 2, 3, and 4 (Relative and Indexed Files)

25. The record to be made available by the READ statement is determined as follows:
- a. *If the file position indicator was positioned by the execution of an OPEN statement, and the PREVIOUS option is specified, the AT END condition occurs. Otherwise, if the file position indicator was positioned by the execution of the START or OPEN statement and the record is still accessible through the path indicated by the file position indicator, the record pointed to by the file position indicator is made available. If the record is no longer accessible, possibly caused by deletion of the record for a relative file or by a change in an alternate key for an indexed file, the file position indicator is updated to point to the next record. If the PREVIOUS option is specified, the file position indicator points to the previous existing record within the established key of reference.* **MF**
 - b. *If the file position indicator was positioned by the execution of a previous READ statement, the file position indicator is updated to point to the next or the previous existing record in the file, if the PREVIOUS option is specified.* **MF**
26. *If the lock mode is AUTOMATIC with either single or multiple record locking and the referenced file is opened I-O, then unless the WITH NO LOCK phrase is specified, the run-unit acquires a record lock for the record retrieved by the successful execution of the READ statement. To read past a locked record, the file position indicator should be updated using the START statement.* **MF**
27. *If the lock mode is MANUAL with single record locking and the referenced file is opened I-O, the run-unit acquires a record lock on the record only if the WITH LOCK phrase is specified. A simple READ statement does not acquire a record lock. To read past a locked record, the file position indicator should be updated using the START statement.* **MF**

-
28. *If the lock mode is MANUAL with multiple record locking and the referenced file is opened I-O, the run-unit acquires a lock on the record only if the WITH KEPT LOCK phrase is specified. A simple READ statement does not acquire a record lock. To read past a locked record, the file position indicator should be updated using the START statement.* MF

Formats 2 and 3 (Relative Files)

29. If the RELATIVE KEY phrase is specified, the execution of a READ statement updates the contents of the RELATIVE KEY data item so it contains the relative record number of the record made available.
30. The execution of a READ statement sets the file position indicator to the record whose relative record number is contained in the data item named in the RELATIVE KEY phrase. If the file does not contain such a record, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. Refer to "INVALID KEY Condition" on page 8-11.

Format 4 (Indexed Files)

31. For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they are released by execution of WRITE statements, or by execution of REWRITE statements which create such duplicate values.
32. If the KEY phrase is not specified, the prime record key is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 or Format 2 READ statements for the file.
33. For an indexed file, if the KEY phrase is specified, data-name-1 or *split-key-name-1* is established as the key of reference for this retrieval. MF
If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 or Format 2 READ statements until a different key of reference is established.
34. Execution of a Format 4 READ statement causes the value of the key of reference to be compared with the value contained in the corresponding data item of the stored records in the file, until the first record having an equal value is found. The file position indicator points to this record which is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. Refer to "INVALID KEY Condition" on page 8-11.

Examples

The following examples show the READ statement:

```
READ PAYROLL-FILE INTO PAYROLL-WORK-AREA.
```

```
READ PAYROLL-FILE
  AT END PERFORM CALCULATE-TOTAL
  CLOSE PAYROLL-FILE.
```

```
READ PAYROLL-FILE
  KEY IS EMPLOYEE-ID
  INVALID KEY PERFORM ERROR-ID-ROUTINE
  NOT INVALID KEY PERFORM UPDATE-PAYROLL-FILE
END-READ.
```

REWRITE Statement

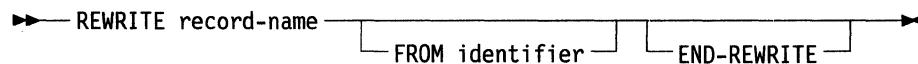
Function

The REWRITE statement logically replaces a record existing in a disk file.

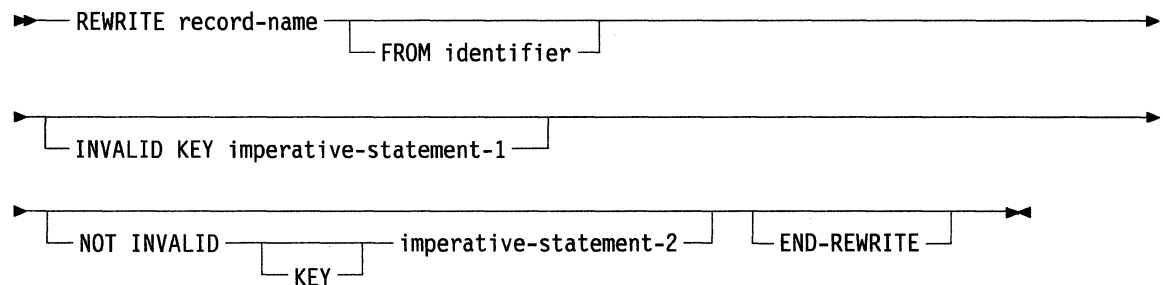
General Format

The following figures show the general format of the REWRITE statement:

Format 1 (Sequential Files)



Format 2 (Relative and Indexed Files)



Syntax Rules

The following syntax rules apply to the REWRITE statement:

All Formats (All Files)

1. Record name and identifier must not refer to the same memory area.
2. Record name is the name of a logical record in the FILE SECTION of the Data Division and may be qualified.

DBCS Support

3. FROM identifier may be a USAGE DISPLAY-1 (DBCS) item.

End of DBCS Support

Format 2 (Relative And Indexed Files)

4. The INVALID KEY phrase must not be specified for a REWRITE statement which references a sequential file or a file in sequential access mode. *However, it can be specified for an indexed file in sequential access mode.* OSVS VSC2

-
5. The **INVALID KEY** phrase must be specified in the **REWRITE** statement for files in the random or dynamic access mode for which an appropriate **USE** procedure is not specified.

General Rules

The following general rules apply to the **REWRITE** statement:

All Formats (All Files)

1. The file associated with record name must be a disk file and must be open in the I-O mode at the time of execution of this statement. Refer to "OPEN Statement" on page 8-62.
2. For files in the sequential access mode, the last input/output statement executed for the associated file prior to the execution of the **REWRITE** statement must have been a successfully executed **READ** statement. The operating system logically replaces the record accessed by the **READ** statement.
3. The number of character positions in the record referenced by record name must be equal to the number of character positions in the record being replaced.
4. The execution of a **REWRITE** statement with the **FROM** phrase is equivalent to the execution of:

MOVE identifier **TO** record-name

followed by the execution of the same **REWRITE** statement without the **FROM** phrase. The contents of the record area prior to the execution of the implicit **MOVE** statement have no effect on the execution of the **REWRITE** statement.

5. The file position indicator is not affected by the execution of a **REWRITE** statement.
6. The execution of the **REWRITE** statement causes the value of the **FILE STATUS** data item associated with the file to be updated. Refer to "I-O Status" on page 8-8.
7. The **END-REWRITE** phrase delimits the scope of the **REWRITE** statement.
8. *A **REWRITE** will not be successful if any other run-unit holds a lock on the record to be rewritten.* **MF**

Format 1 (Sequential Files)

9. The logical record released by a successful execution of the **REWRITE** statement is no longer available in the record area unless the associated file is named in a **SAME RECORD AREA** clause. In this case, the record is still available to the program in the record area as a record of this file, and also as a record of other files named in the **SAME RECORD AREA** clause.
10. The **REWRITE** statement cannot be used with line-sequential files. *The **REWRITE** statement can be used with line-sequential files only if the rewritten record is the same length as the record that is read. When a record is read, null and tab characters in the record may be expanded and this makes the record longer.* **MF**

Format 2 (Relative And Indexed Files)

11. The logical record released by a successful execution of the **REWRITE** statement is no longer available in the record area unless the associated file is saved in a **SAME RECORD AREA** clause. If so, the logical record is available to the program as a record of other files appearing in the same **SAME RECORD AREA** clause as the associated I-O file, as well as to the file associated with record name.

-
12. *If a file has multiple record locking, the REWRITE statement can be made to acquire a record lock by use of an AIX VS COBOL system directive. Refer to the User's Guide for details of these directives.* MF

Format 2 (Relative Files)

13. For a file accessed in either random or dynamic access mode, the operating system logically replaces the record specified by the contents of the RELATIVE KEY data item associated with the file. If the file does not contain the record specified by the key, the INVALID KEY condition exists. Refer to "INVALID KEY Condition" on page 8-11. If the INVALID KEY condition exists, the updating operation does not take place and the data in the record area is unaffected.

Format 2 (Indexed Files)

14. For a file in the sequential access mode, the record to be replaced is specified by the value contained in the prime record key. When the REWRITE statement is executed, the value contained in the prime record data item of the record to be replaced must be equal to the value of the prime record key of the last record read from this file.
15. For a file in the random or dynamic access mode, the record to be replaced is specified by the prime record key data item.
16. The contents of alternative record key data items of the record being rewritten may differ from those in the record being replaced. The operating system utilizes the content of the record key data items during the execution of the REWRITE statement in such a way that subsequent access of the record may be made based upon any of those specified record keys.
17. The INVALID KEY condition exists when:
- a. *The access mode is sequential and the value contained in the prime record key data item of the record to be replaced is not equal to the value of the prime record key of the last record read from this file.* OSVS VSC2
 - b. The value contained in the prime record key data item does not equal that of any record stored in the file.
 - c. The value contained in an alternate record key data item for which a DUPLICATES clause has not been specified is equal to that of a record already stored in the file.
- When the INVALID KEY condition exists, the updating operation does not take place and the data in the record area is unaffected. Refer to "INVALID KEY Condition" on page 8-11.

Example

The following example shows the REWRITE statement:

```
OPEN I-O ACCT-PAYABLE.  
  :  
  :  
READ ACCT-PAYABLE RECORD INTO ACCT-WORK.  
  :  
  :  
  (Change contents of ACCT-WORK if needed.)  
  :  
  :  
REWRITE ACCT-PAYABLE-RECORD FROM ACCT-WORK.
```

The last INPUT-OUTPUT statement executed for the associated file prior to the execution of the REWRITE must have been a successfully executed READ statement.

START Statement

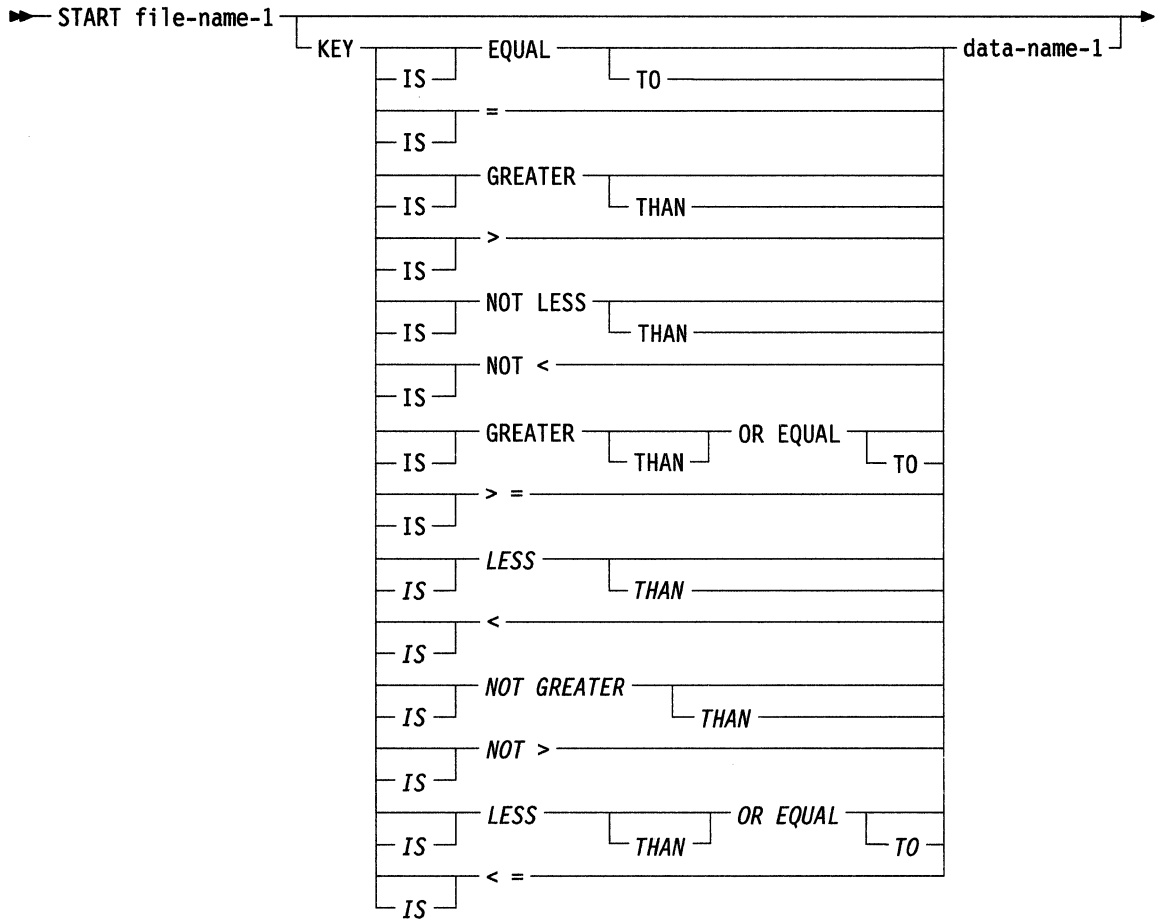
Function

The START statement provides a basis for logical positioning within a relative or indexed file for subsequent retrieval of records. This statement is not available for files with sequential organization.

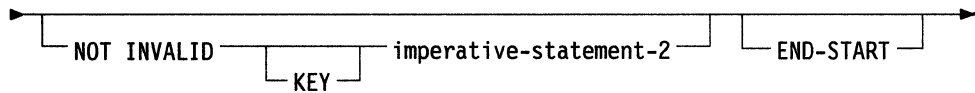
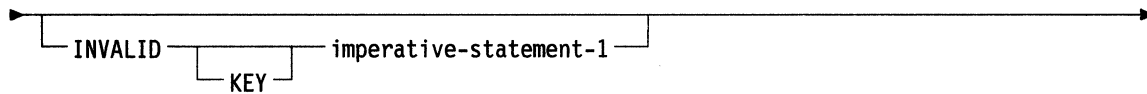
General Format

The following figures show the general formats of the START statement:

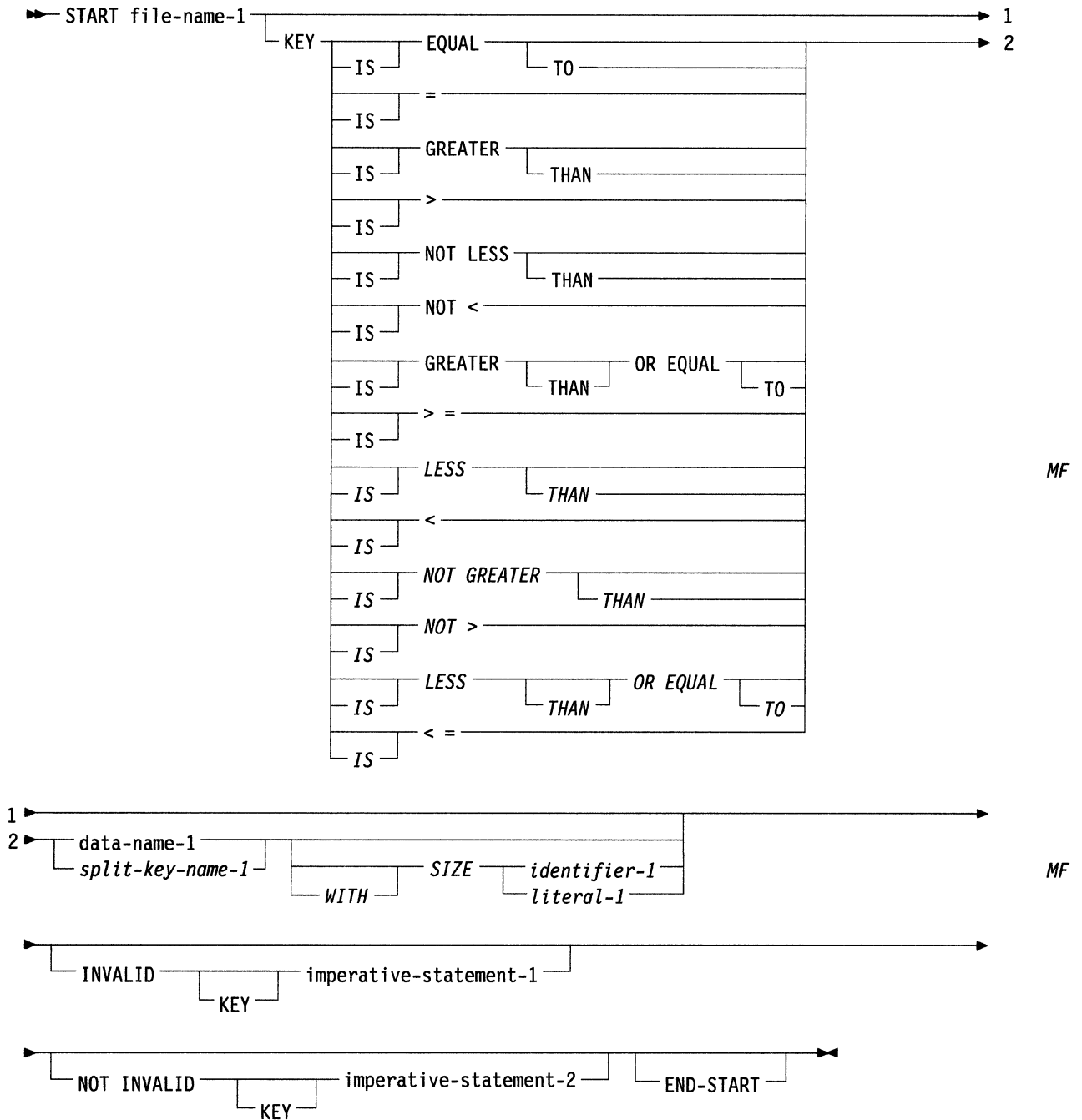
Format 1 (Relative Files)



MF



Format 2 (Indexed Files)



Syntax Rules

The following syntax rules apply to the START statement:

All Formats (Relative and Indexed Files)

1. The file name must be the name of a relative or indexed file.
2. The file name must be the name of a file with sequential or dynamic access.
3. Data-name-1 may be qualified.
4. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file name.

Format 1 (Relative Files)

5. If the KEY phrase is specified, data-name-1 must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.

Format 2 (Indexed Files)

6. If the KEY phrase is specified, data-name-1 may reference a data item specified as a record key associated with the file name. Data-name-1 may also reference any data item of category alphanumeric, subordinate to the data item specified as a record key, associated with file name in which the leftmost character position corresponds to the leftmost character position of that record key data item.
7. *split-key-name-1 may reference one or more data items and is specified as a record key associated with file name.* MF
8. *WITH SIZE specifies the number of characters in the key to be used in the positioning process.* MF
9. *identifier-1 must be the name of an elementary integer data item when used with the WITH SIZE phrase.* MF

General Rules

The following general rules apply to the START statement:

All Formats (Relative and Indexed Files)

1. The file name must be open in the INPUT or I-O mode at the time that the START statement is executed. Refer to "OPEN Statement" on page 8-62.
2. If the KEY phrase is not specified, the relational operation IS EQUAL TO is implied.
3. The execution of the START statement causes the value of any FILE STATUS data item associated with file name to be updated. Refer to "I-O Status" on page 8-8.
4. *The START statement neither acquires nor detects a record lock.* MF

Format 1 (Relative Files)

5. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file name and a data item as specified in general rule 6 on page 8-83.
 - a. *If the relational operation specifies that the key must be greater than, or greater than or equal to the data item, then the file position indicator points to the first logical record existing in the file containing the key satisfying the comparison.* MF

-
- b. *If the relational operator specifies that the key must be less than, or less than or equal to the data item, then the file position indicator points to the last logical record existing in the file containing the key satisfying the comparison.* MF
 - c. If the comparison is not satisfied by any record in the file, an INVALID KEY exists, the execution of the START statement is unsuccessful, and the position of the file position indicator is undefined. Refer to "INVALID KEY Condition" on page 8-11.
6. The comparison described in 5 on page 8-82 uses the data item referenced by the RELATIVE KEY clause associated with file name. A RELATIVE KEY clause must be associated with a file name.

Format 2 (Indexed Files)

7. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file name and a data item as specified in general rule 8. If a file name references an indexed file and the operands are of unequal size, comparison proceeds as though the longer one were truncated on the right, making its length equal to that of the shorter. All other nonnumeric comparison rules apply, except that the presence of the PROGRAM COLLATING SEQUENCE clause has no effect on the comparison. Refer to "Comparison of Nonnumeric Operands" on page 7-12.
 - a. *If the relational operator specifies that the key must be greater than or equal to the data item, then the file position indicator points to the first logical record currently existing in the file whose key satisfies the comparison.* MF
 - b. *If the relational operator specifies that the key must be less than, or less than or equal to the data item, then the file position indicator points to the last logical record currently existing in the file whose key satisfies the comparison.* MF
 - c. If the comparison is not satisfied by any record in the file, an INVALID KEY exists, the execution of the START statement is unsuccessful, and the position of the file position indicator is undefined. Refer to "INVALID KEY Condition" on page 8-11.
8. If the KEY phrase is specified, the comparison described in general rule 7 uses the data item referenced by data-name-1.
9. If the operands are of unequal size, comparison proceeds as though the longer one were truncated on the right, making its length equal to that of the shorter. All other nonnumeric comparison rules apply except that the presence of the PROGRAM COLLATING SEQUENCE clause has no effect on the comparison. Refer to "Comparison of Nonnumeric Operands" on page 7-12.
10. *If the WITH SIZE phrase is specified, the relational operator in the KEY phrase is ignored.* MF
11. If the KEY phrase is not specified, the comparison described in general rule 7 uses the data item referenced in the RECORD KEY clause associated with file name.
12. Upon the successful execution of the START statement, a key of reference is established and used in subsequent Format 1 or Format 3 READ statements as follows (refer to "READ Statement" on page 8-68):
 - a. If the KEY phrase is not specified, the prime record key specified for file name becomes the key of reference.
 - b. If the KEY phrase is specified, and data-name-1 or *split-key-name-1* is specified as a record key for file name, that record key becomes the key of reference. MF

-
- c. If the **KEY** phrase is specified, and *data-name-1* or *split-key-name-1* is not specified as a record key for file name, the record key with a leftmost character position corresponding to the leftmost character position of the data item specified by *data-name-1* or *split-key-name-1* becomes the key of reference. *MF*

13. If the execution of the **START** statement is not successful, the key of reference is undefined.

Examples

The following examples show the **START** statement.

```
MOVE "DOE" TO START-NAME.  
START AUTHOR-FILE KEY >= START-NAME  
  NOT INVALID KEY  
    PERFORM PRINT-AUTHORS  
END-START.
```

```
START TRANS-HISTORY KEY >= START-DATE.
```

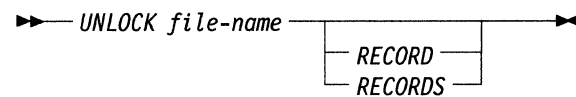
UNLOCK Statement

Function

The UNLOCK statement releases all record locks acquired by the run-unit on a named file. *MF*

General Format

The following figure shows the general format of the UNLOCK statement: *MF*



General Rules

The following general rules apply to the UNLOCK statement: *MF*

- 1. File name must occur in the SELECT statement of the FILE CONTROL entry.*
- 2. The file referenced by file name must already be opened with the OPEN statement.*

USE Statement

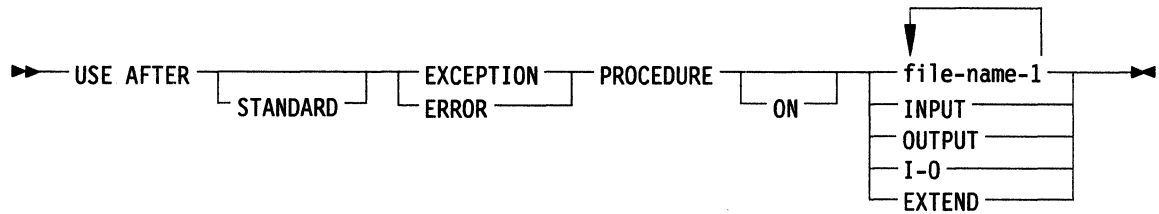
Function

The USE statement specifies procedures for input-output error handling in addition to the standard procedures provided by the input-output control system. An input-output error on a file invokes the USE procedure only if the file has a file status item.

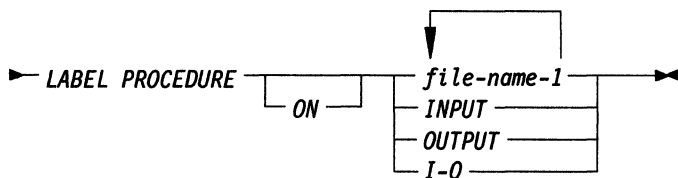
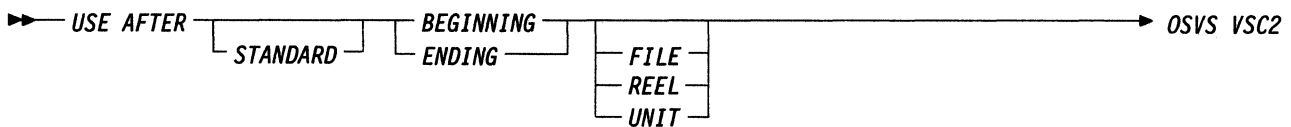
General Format

The following figures show the general format of the USE statement:

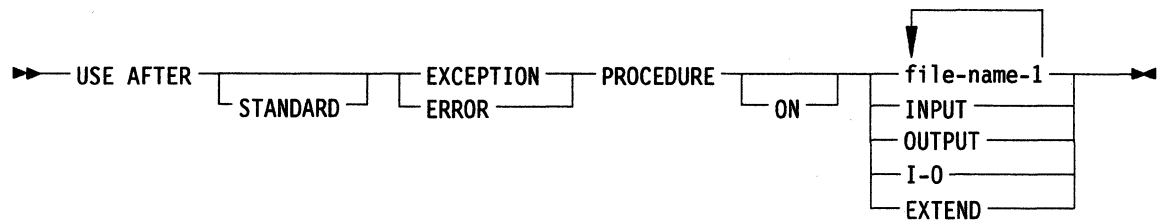
Format 1 (Sequential Files)



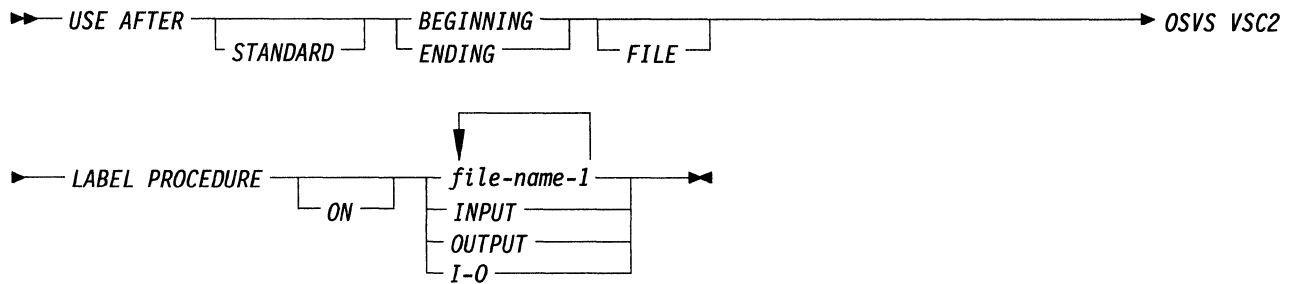
Format 2 (Sequential Files)



Format 3 (Relative and Indexed Files)



Format 4 (Relative and Indexed Files)



Syntax Rules

The following syntax rules apply to the USE statement:

All Formats (All Files)

1. Formats 1 and 3 are the **ERROR** declarative, and *Formats 2 and 4 are OSVS VSC2 the LABEL declarative.*
2. A USE statement, when present, must immediately follow a section header in the declaratives section and must be followed by a period, followed by a space. The remainder of the section must consist of zero, one, or more procedural paragraphs defining the procedure to be used.
3. The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.
4. The same file name can appear in a different specific arrangement of the format. Appearance of a file name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.
5. The words **ERROR** and **EXCEPTION** are synonymous and may be used interchangeably.
6. The files implicitly or explicitly referenced in a USE statement need not all have the same organization or access.

Formats 2 And 4 (All Files)

7. *If both BEGINNING and ENDING are omitted, the effect is as though OSVS VSC2 both BEGINNING and ENDING had been specified.*

Format 2 (Sequential Files)

8. *REEL and UNIT are treated as equivalent. OSVS VSC2*
9. *If both FILE and REEL or UNIT are omitted, the effect is as though OSVS VSC2 both had been specified.*
10. *Any file name and any one OPEN mode may appear in, at most, one OSVS VSC2 declarative for each of the possible combinations of BEGINNING|ENDING and FILE|REEL as shown below:*

<i>BEGINNING</i>	<i>FILE</i>
<i>BEGINNING</i>	<i>REEL OR UNIT</i>
<i>ENDING</i>	<i>FILE</i>
<i>ENDING</i>	<i>REEL OR UNIT</i>

General Rules

The following general rules apply to the USE statement:

All Formats (All Files)

1. After execution of a USE procedure, control is returned to the invoking routine.
2. Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure names that appear in the declarative portion, except that PERFORM statements may refer to a USE statement.
3. The execution of any statement causing the execution of a USE procedure previously invoked, but with control not yet returned to the invoking routine, is not allowed within a USE procedure.

Formats 1 And 3 (All Files)

4. The designated procedures are executed by the input-output system after completing the standard input-output error routine. The procedures are also executed upon recognition of the AT END condition, when the AT END phrase has not been specified in the input-output statement.
5. When file-name-1 is specified explicitly, no other USE statement applies to file-name-1.

Formats 2 And 4 (All Files)

6. *This declarative is never executed unless by an explicit PERFORM statement in the nondeclarative portion of the Procedure Division.* OSVS VSC2
7. *The statement GO TO MORE LABELS is used as a simple jump to the start of the declarative procedure in which it appears.*

Example

The following example shows the USE statement:

```
PROCEDURE DIVISION.  
DECLARATIVES.  
ERROR-HANDLING SECTION.  
    USE AFTER STANDARD ERROR PROCEDURE ON INPUT.  
    :  
    :  
END DECLARATIVES.
```

WRITE Statement

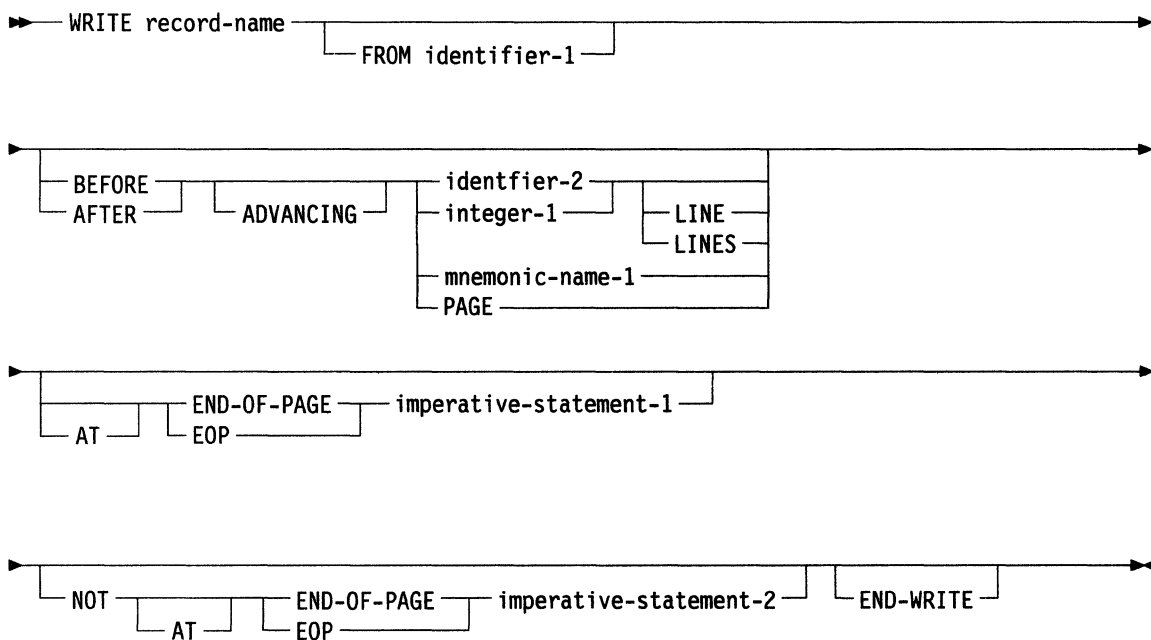
Function

The WRITE statement releases a logical record for an output or input-output file. For sequential files it can also be used for vertical positioning of lines within a logical page.

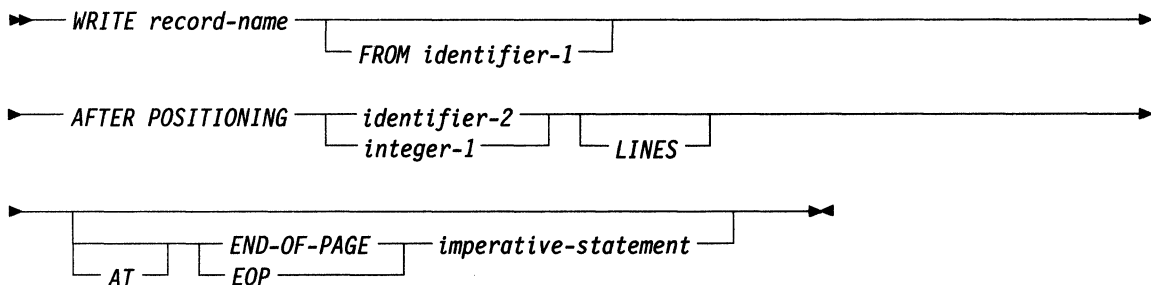
General Format

The following figures show the general format of the WRITE statement:

Format 1 (Sequential Files)

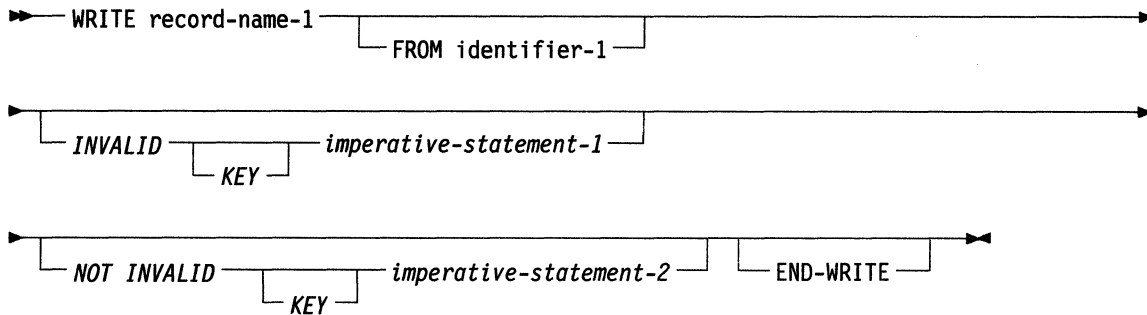


Format 2 (Sequential Files)



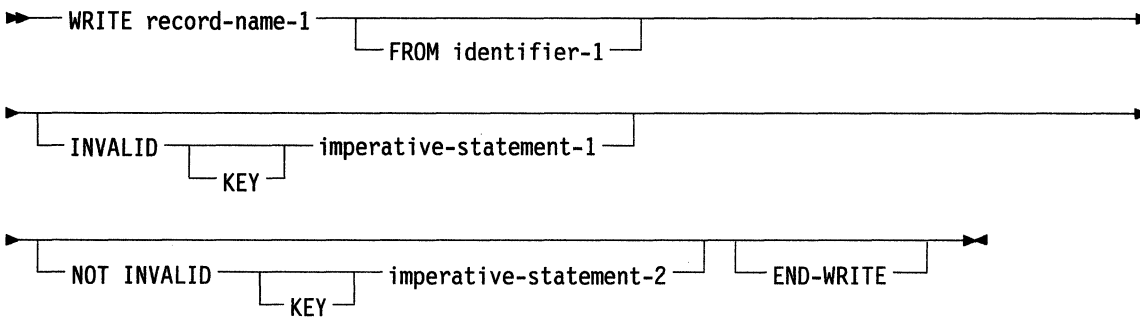
OSVS

Format 3 (Sequential Files)



MF

Format 4 (Relative and Indexed Files)



Syntax Rules

The following syntax rules apply to the `WRITE` statement:

All Formats (All Files)

1. Record name and identifier-1 must not reference the same memory area.
2. The record name is the name of a logical record in the `FILE SECTION` of the Data Division and may be qualified.

DBCS Support

3. identifier-1 may be a `USAGE DISPLAY-1` (DBCS) item.

End of DBCS Support

Format 1 (Sequential Files)

4. When the mnemonic name associated with `TAB` is specified, the result is to cause the paper to throw to the standard vertical tabulation position. A user-defined mnemonic name can be used instead of `TAB` or `FORMFEED` if they are associated in the `SPECIAL-NAMES` paragraph. Refer to "SPECIAL-NAMES Paragraph" on page 5-8.
5. When identifier-2 is used in the `ADVANCING` phrase, it must be the name of an elementary integer data item.
6. Integer-1, or the value of the data item referenced by identifier-2, may be zero.

-
7. If the END-OF-PAGE phrase is specified, the LINAGE clause must be specified in the file description entry for the associated file.
 8. The words END-OF-PAGE and EOP are equivalent.
 9. The ADVANCING mnemonic name phrase cannot be specified when writing a record to a file whose file description entry contains the LINAGE clause.

Format 2 (Sequential Files)

10. *This format cannot be specified when writing a record to a file whose file description entry contains the LINAGE clause.* OSVS
11. *If this format of the WRITE statement is used for writing to a given file, then every WRITE statement used for that file should be in this format.* OSVS
12. *In the AFTER POSITIONING phrase, identifier-2 must be defined as a single character alphanumeric item. Refer to general rule 14 on page 8-93 for its possible values.* OSVS

Format 4 (Relative and Indexed Files)

13. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

General Rules

The following general rules apply to the WRITE statement:

All Formats (All Files)

1. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of:
 - a. The statement:
MOVE identifier-1 TO record name
according to the rules specified for the MOVE statement, followed by
 - b. The same WRITE statement without the FROM phrase.The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.
After execution of the 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 may not be. Refer to general rules 8 and 17.
2. The file position indicator is unaffected by the execution of a WRITE statement.
3. The execution of the WRITE statement causes the value of the FILE STATUS data item associated with the file to be updated. Refer to "I-O Status" on page 8-8.
4. The maximum record size for a file is established at the time the file is created and cannot be changed.
5. The number of character positions on a mass storage device to store a logical record may or may not be equal to the number of character positions defined by the logical description of that record in the program.
6. The execution of the WRITE statement releases a logical record to the operating system.

Formats 1, 2, And 3 (Sequential Files)

7. The associated file must be open in the OUTPUT or EXTEND mode at the time of the execution of this statement. Refer to "OPEN Statement" on page 8-62.
8. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement was unsuccessful due to a boundary violation.

The logical record is also available to the program as a record of other files referenced in the SAME RECORD AREA clause as the associated output file, as well as to the file associated with record name.

Format 1 (Sequential Files)

9. When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists and the contents of the record area are unaffected. The following action takes place:
 - a. The value of the FILE STATUS data item of the associated file is set to a value indicating a boundary violation. Refer to "I-O Status" on page 8-8.
 - b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file, that declarative procedure is then executed.
 - c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the result is undefined.
10. After the recognition of an end of unit of an output file contained on more than one physical reel or unit, the WRITE statement performs the following operations:
 - a. The standard ending reel or unit procedure
 - b. The reel or unit swap
 - c. The standard beginning of reel or unit label procedure.
11. Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page.
 - a. With ORGANIZATION RECORD-SEQUENTIAL, if the ADVANCING phrase is not used, automatic advancing is provided when output is directed to a list device to act as if the user had specified AFTER ADVANCING 1 LINE. One exception to this is if you WRITE a sequential file to a list device via a pipe, automatic line advancing is not provided. Instead, the file behaves as if you had specified a WRITE statement only. Therefore, you must either explicitly WRITE AFTER ADVANCING 1 LINE, or specify the printer in a SELECT ASSIGN statement. If the ADVANCING phrase is used, advancing is provided as follows:
 - 1) If identifier-2 is specified, the representation of the printed page is advanced the number of lines equal to the current value associated with identifier-2.
 - 2) If integer is specified, the representation of the printed page is advanced in the number of lines equal to the value of integer.
 - 3) If mnemonic-name is specified, the representation of the printed page is advanced according to the rules specified by the implementer for the hardware device.
 - 4) If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to general rules 11a1, 11a2, and 11a3.
 - 5) If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to general rules 11a1, 11a2, and 11a3.
 - 6) If PAGE is specified, the record is presented on the logical page before or after the device (depending on the phrase used) is repositioned to the next logical page. If the record to be written is associated with a file containing a LINAGE clause in the file description entry, the repositioning is to the first line that can be written on the next logical page as specified in the LINAGE clause.

-
- b. With ORGANIZATION LINE-SEQUENTIAL, if the ADVANCING phrase is not used, automatic advancing of one line is provided to act in accordance with the convention of your operating system text editor (usually as if the user had specified BEFORE ADVANCING 1 LINE).

If the ADVANCING phrase is used, advancing is provided according to general rule 11a.

12. If the logical end of the representation of the printed page is reached during the execution of a WRITE statement with the END-OF-PAGE phrase, the imperative statement specified in the END-OF-PAGE phrase is executed. The logical end is specified in the LINAGE clause associated with record name.
13. An end-of-page condition occurs whenever a given WRITE statement with the END-OF-PAGE phrase is executed and the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by integer-2 or the data item referenced by data-name-2 of the LINAGE clause. In this case, the WRITE statement is executed and then the imperative statement in the END-OF-PAGE phrase is executed.

An automatic page overflow condition occurs whenever the execution of a given WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body. This occurs when the execution of a WRITE statement causes the LINAGE-COUNTER to exceed the value specified by integer-1 or the data item referenced by data-name-1 of the LINAGE clause. In this case, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the first line that can be written on the next logical page as specified in the LINAGE clause. The imperative statement in the END-OF-PAGE clause, if specified, is executed after the record is written and the device has been repositioned.

If integer-2 or data-name-2 of the LINAGE clause is not specified, no end-of-page condition distinct from the page overflow condition is detected. In this case, the end-of-page condition and page overflow condition occur simultaneously.

If integer-2 or data-name-2 of the LINAGE clause is specified, but the execution of a given WRITE statement would cause LINAGE-COUNTER to simultaneously exceed the value of both integer-2 or the data item referenced by data-name-2 and integer-1 or the data item referenced by data-name-1, then the operation proceeds as if integer-2 or data-name-2 had not been specified.

Format 2 (Sequential Files)

14. *When the AFTER POSITIONING phrase is used in a WRITE statement, the system moves a suitable character into the first position of the record before it is written to the file. This first character position must be reserved by the user for this purpose. If the identifier-2 option is used, then the character moved into the output record is simply the value held by identifier-2 and should be one of the values in Table 8-3 on page 8-95.* OSVS

If the integer-1 option is used, then the character placed in the output record is determined by the values in Table 8-4 on page 8-95.

The implicit move always takes place according to the above rules, but actions based on the given interpretations are system-dependent. Refer to the User's Guide.

15. The END-OF-PAGE phrase, if specified, is treated as documentary and is never executed.

Format 3 (Sequential Files)

16. When an attempt is made to write beyond the externally defined boundaries of a sequential file, an **INVALID KEY** condition occurs. When the **INVALID KEY** condition is recognized, the execution of the **WRITE** statement is unsuccessful, the contents of the record area are unaffected, and the **FILE STATUS** data item, if any, of the associated file is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules stated in "**INVALID KEY Condition**" on page 8-11. Refer also to "**I-O Status**" on page 8-8. **MF**

Format 4 (Relative And Indexed Files)

17. The associated file must be open in the **OUTPUT** or **I-O** mode at the time this statement is executed. Refer to "**OPEN Statement**" on page 8-62.
18. The logical record released by the execution of the **WRITE** statement is no longer available in the record area unless the associated file is named in a **SAME RECORD AREA** clause or the execution of the **WRITE** statement is unsuccessful due to an **INVALID KEY** condition.

The logical record is available to the program and file associated with record name, as a record of other files referenced in the same **SAME RECORD AREA** clause as the associated output file.

19. When the **INVALID KEY** condition is recognized, the execution of the **WRITE** statement is unsuccessful, the contents of the record area are unaffected, and the **FILE STATUS** data item of the associated file, if any, is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules stated in "**INVALID KEY Condition**" on page 8-11. Refer also to "**I-O Status**" on page 8-8.
20. If the file has multiple record locking, the **WRITE** statement can be made to acquire a record lock by use of an **AIX VS COBOL** system directive. **MF**
Refer to the *User's Guide* for details of directives.

Format 4 (Relative Files)

21. When a file is opened in the output mode, records may be placed into the file by one of the following:
- a. If the access mode is sequential, the **WRITE** statement causes a record to be released to the operating system. The first record has a relative record number of one and subsequent records released have relative record numbers of 2, 3, 4, If the **RELATIVE KEY** data item has been specified in the file control entry for the associated file, the relative record number of the record just released is placed into the **RELATIVE KEY** data item by the operating system during execution of the **WRITE** statement.
 - b. If the access mode is random or dynamic, prior to the execution of the **WRITE** statement, the value of the **RELATIVE KEY** data item must be initialized in the program with the relative record number or be associated with the record in the record area. The record is then released to the operating system by execution of the **WRITE** statement.
22. When a file is opened in the **I-O** mode and the access mode is random or dynamic, records are to be inserted in the associated file. The value of the **RELATIVE KEY** data item must be initialized by the program with the relative record number to be associated with the record in the record area. Execution of a **WRITE** statement then causes the contents of the record area to be released by the operating system.
23. The **INVALID KEY** condition exists under the following circumstances:
- a. When the access mode is random or dynamic, and the **RELATIVE KEY** data item specifies a record that already exists in the file.
 - b. When an attempt is made to write beyond the externally defined boundaries of the file.

Format 4 (Indexed Files)

24. Execution of the WRITE statement causes the contents of the record area to be released. The operating system utilizes the contents of the record keys so the user can access the record using any of the specified record keys.
25. The value of the prime record key must be unique within the records in the file.
26. The data item specified as the prime record key must be set by the program to the desired value prior to the execution of the WRITE statement.
27. If sequential access mode is specified for the file, records must be released to the operating system in ascending order of prime record key values.
28. If random or dynamic access mode is specified, records may be released to the operating system in any program specified order.
29. When the ALTERNATE RECORD KEY clause is specified in the file control entry for an indexed file, the value of the alternate record key may be nonunique only if the DUPLICATES phrase is specified for that data item. In this case, the operating system provides storage of records such that when records are accessed sequentially, the order of retrieval is the order in which they are released to the operating system.
30. The INVALID KEY condition exists under the following circumstances:
 - a. When sequential access mode is specified for a file opened in the output mode, and for the value of the prime record key of the previous record.
 - b. When the file is opened in the output or I-O mode, and the value of the prime record key is equal to the value of a prime record key of a record already existing in the file.
 - c. When the file is opened in the output or I-O mode, and the value of an alternate record key for which duplicates are not allowed equals the corresponding data item of a record already existing in the file.
 - d. When an attempt is made to write beyond the externally defined boundaries of the file.

Table 8-3. AFTER POSITIONING Phrase with identifier-2

Identifier-2	Interpretation
(space)	Single spacing
0	Double spacing
-	Triple spacing
+	Suppress spacing
1-9	Skip to channel 1-9, respectively
A, B, C	Skip to channel 10, 11, 12, respectively
V, W	Pocket select 1 or 2

Table 8-4. AFTER POSITIONING Phrase with integer-1

Integer-1	Output Character	Interpretation
0	1	Skip to channel 1
1	(space)	Single space
2	0	Double spacing
3	-	Triple spacing

Example

The following example shows the WRITE statement:

```
WRITE EMPL-RECORD.
```

```
WRITE EMPL-RECORD FROM EMPL-WORK  
  INVALID KEY PERFORM ERROR-WRITE.
```

```
WRITE OUTPUT-LINE-REC AFTER ADVANCING 2 LINES.
```

```
WRITE ACCT-PAYABLE-REC  
  INVALID KEY DISPLAY "BAD RECORD KEY"  
  :  
  :  
  NOT INVALID KEY  
  :  
  :  
END-WRITE.
```

Chapter 9. COBOL Source Library

Contents

About This Chapter	9-3
Introduction	9-4
COPY Statement	9-5
Function	9-5
General Format	9-5
Syntax Rules	9-5
General Rules	9-6
Example 1	9-8
Example 2	9-8
REPLACE Statement	9-10
Function	9-10
General Format	9-10
Syntax Rules	9-10
General Rules	9-10

About This Chapter

This chapter describes a facility in COBOL which may be used to ease coding and to support standardized coding. This facility is in the Source Library Module, which provides two compiler-directing statements, COPY and REPLACE, to copy prewritten text from a source user-library to a COBOL program or to globally replace source program text in the program.

Introduction

The library module provides a capability for specifying text to be copied from a source user library file. This is usually created using any suitable source text editor. It also provides a capability for replacing text in the source program.

IBM AIX VS COBOL libraries consist of fixed-disk files containing source material to be made available to the COBOL system. The effect of the interpretation of the COPY statement is to insert text into the source program, where it is treated by the COBOL system as part of the source program. All occurrences of a given literal, identifier, word, or group of words in the library text can be replaced with alternate text during the copy process. The library module also makes more than one COBOL library available at the time the object code is created.

The effect of the REPLACE statement is to substitute new text for text appearing in the source program and have the new text treated by the AIX VS COBOL system as part of the source program.

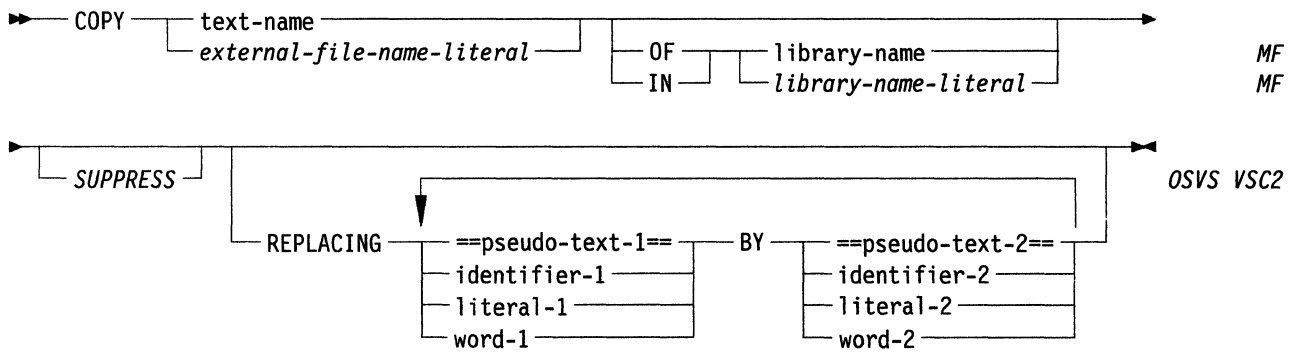
COPY Statement

Function

The COPY statement incorporates text into an AIX VS COBOL source program.

General Format

The following figure shows the general format of the COPY statement:



Syntax Rules

The following syntax rules apply to the COPY statement:

1. If more than one COBOL library is available while the source is passed through the COBOL system, text name must be qualified by the library name identifying the COBOL library in which the text associated with text name resides. Refer to the *User's Guide*.
2. The COPY statement must be preceded by a space and terminated by the separator period.
3. pseudo-text-1 must not be null, nor may it consist solely of the character space(s), nor may it consist solely of comment lines.
4. pseudo-text-2 may be null.
5. Character strings within pseudo-text-1 and pseudo-text-2 may be continued. However, both characters of a pseudo-text delimiter must be on the same line.
6. word-1 or word-2 may be any single COBOL word.
7. A COPY statement may occur in the source program anywhere a character string or a separator may occur except that a COPY statement may not occur within a COPY statement. A COPY statement in a library text must not specify the same library text.

Nested copy statements are allowed. However, recursive COPY statements (where a library-text is referred to by a COPY statement within it) are not permitted.

OSVS VSC2

-
- 8. text-name defines a unique external file name which conforms to the rules for user-defined words (lowercase is converted to uppercase). *The external-file-name-literal is an alphanumeric literal that conforms to the operating system rules for file names. It must be specified within enclosing quotes.* MF
 - 9. *The library-name-literal is an alphanumeric literal that conforms either to the operating system rules for file names or to the operating system rules for device identifiers. It may be specified with or without enclosing quotes.* MF
 - 10. *The SUPPRESS phrase is used to suppress printing the contents of the copy member on the source listing but is documentary only.* OSVS VSC2
 - 11. If the word COPY appears in a comment-entry or in the place where a comment-entry may appear, it is considered part of the comment-entry.
 - 12. *It is not permitted for the word COPY in a COPY statement appearing in a library text to be continued onto a new line.* OSVS VSC2

DBCS Support

- 13. The REPLACING operands may be USAGE DISPLAY-1 items, which are Double-Byte Character Set (DBCS) items.

End of DBCS Support

General Rules

The following rules apply to the COPY statement:

- 1. The compilation of a source program containing COPY statement is logically equivalent to processing all COPY statements prior to the processing of the resulting source program.
- 2. 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 reserved word COPY and ending with the punctuation character period. *Refer to rule 13 on page 9-8 for OLDCOPY alternative processing.* OSVS
- 3. If the REPLACING phrase is not specified, the library text is copied unchanged.
If the REPLACING phrase is specified, the library text is copied and each properly matched occurrence of pseudo-text-1, identifier-1, literal-1, and word-1 in the library text is replaced by the corresponding pseudo-text-2, identifier-2, literal-2, or word-2.
- 4. For purposes of matching, identifier-1, literal-1, and word-1 are treated as pseudo-text containing only identifier-1, literal-1, or word-1, respectively.

-
5. The comparison operation to determine text replacement occurs in the following manner:

Any separator comma, semicolon, and/or space(s) preceding the leftmost library text word is copied into the source program. Starting with the leftmost library text word and the first pseudo-text-1, the entire REPLACING phrase operand preceding the reserved word BY is compared to an equivalent number of contiguous library text words. pseudo-text-1, identifier-1, literal-1, or word-1 match the library text if the ordered sequence of text words forming pseudo-text-1, identifier-1, literal-1, or word-1 is equal, character for character, to the ordered sequence of library text words. For purposes of matching, each occurrence of a separator comma or semicolon in pseudo-text-1 or in the library text is considered to be a single space except when pseudo-text-1 consists solely of either a separator comma or semicolon. In this case, it is considered to be a text word. Each sequence of one or more space separators is considered to be a single space.

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

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

Whenever a match occurs between pseudo-text-1, identifier-1, literal-1, or word-1 and the library text, the corresponding pseudo-text-2, identifier-2, literal-2, or word-2 is placed into the source program. The library text word immediately following the rightmost text word being compared is then considered the leftmost library text word. The comparison cycle starts again with the first pseudo-text-1, identifier-1, literal-1, or word-1 specified in the REPLACING phrase.

The comparison operation continues until the rightmost text word in the library text has either been matched or has been a leftmost library text word and compared through a complete cycle.

6. A comment line occurring in the library text and pseudo-text-1 is interpreted, for purposes of matching, as a single space. Comment lines appearing in pseudo-text-2 and library text are copied into the source program unchanged.
7. Debugging lines are permitted within library text and pseudo-text-2. Debugging lines are not permitted within pseudo-text-1. Text words within a debugging line are treated 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.
8. The text produced as a result of the complete processing of a COPY statement must not contain a COPY statement. *This text may contain a COPY statement provided neither this contained COPY nor the already expanded COPY includes the REPLACING phrase.* VSC2
9. The syntactic correctness of the library text cannot be independently determined. Except for COPY statements, the syntactic correctness of the entire COBOL source program cannot be determined until all COPY statements have been completely processed.
10. Each text word copied from the library is copied so it starts in the same area of the line in the resultant program as it begins in the library. Library text must conform to the rules for COBOL reference format. If additional lines are introduced into the same 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.
11. For purposes of compilation, text words after replacement are placed in the source program according to the rules for reference format as described in Chapter 1, "Introduction."

-
12. If the unit identifier is not explicitly specified, the default drive is used. The default is operating system-dependent and is described in the *User's Guide*.
13. *The COBOL system directive OLD COPY can be set to cause the entire entry to be replaced by the information identified by text name, except that the data-name preceding the COPY statement replaces the corresponding data-name in the text name.* See the following examples. OSVS

Example 1

The following example is performed with the OLD COPY directive set:

Source-file Code

```
01 PRODUCT-CODE COPY COPYPROD.
```

Copy-file "COPYPROD"

```
01 PROD-CD.  
05 ITEM-NAME PIC X(30).  
05 ITEM-NUMBER PIC X(5).
```

Resulting COBOL Code

```
01 PRODUCT-CODE.  
05 ITEM-NAME PIC X(30).  
05 ITEM-NUMBER PIC X(5).
```

Example 2

In the following example, the library entry PAYLIB consists of these Data Division entries:

```
02 B PIC S99.  
02 C PIC S9(5)V99.  
02 D PIC S9999 OCCURS 1 TO 52 TIMES  
DEPENDING ON B OF A.
```

You can use the COPY statement in the Data Division of a program as follows:

```
01 PAYROLL. COPY PAYLIB.
```

In this program, the library entry is then copied. The resulting entry is treated as if it were written as follows:

```
01 PAYROLL.  
02 B PIC S99.  
02 C PIC S9(5)V99.  
02 D PIC S9999 OCCURS 1 TO 52 TIMES  
DEPENDING ON B OF A.
```

To change names within the library entry, use the REPLACING option:

```
01 PAYROLL. COPY PAYLIB REPLACING A BY PAYROLL  
B BY PAY-CODE C BY GROSS-PAY.
```

In this program, the library entry is copied. The resulting entry is treated as if it were written as follows:

```
01  PAYROLL.  
02  PAY-CODE  PIC S99.  
02  GROSS-PAY PIC S9(5)V99.  
02  D        PIC S9999 OCCURS 1 TO 52 TIMES  
                DEPENDING ON PAY-CODE OF PAYROLL.
```

The changes shown in the foregoing examples are made for this program only. The entry as it appears in the library remains unchanged.

REPLACE Statement

Function

The REPLACE statement is used to replace source program text.

General Format

The following figures show the format of the REPLACE statement:

Format 1

Diagram illustrating the Format 1 REPLACE statement. The text is: `REPLACE ==pseudo-text-1== BY ==pseudo-text-2==`. The text is enclosed in a double-headed arrow. A line with a downward-pointing arrowhead connects the end of the first pseudo-text to the start of the second pseudo-text, indicating the replacement.

Format 2

Diagram illustrating the Format 2 REPLACE statement. The text is: `REPLACE OFF`. The text is enclosed in a double-headed arrow.

Syntax Rules

The following syntax rules apply to the REPLACE statement:

1. A REPLACE statement may occur anywhere in the source program where a character string may occur. It must be preceded by a separator period except when it is the first statement in a separate program.
2. A REPLACE statement must be terminated by a separator period.
3. pseudo-text-1 must contain one or more text words.
4. pseudo-text-2 may contain zero, one, or more text words.
5. Character strings within pseudo-text-1 and pseudo-text-2 may be continued.
6. A text word within pseudo-text must be between 1 and 322 characters long.
7. pseudo-text-1 must not consist entirely of a separator comma or a separator semicolon.
8. If the word REPLACE appears in a comment-entry or in the place where a comment-entry may appear, it is considered part of the comment-entry.

General Rules

The following general rules apply to the REPLACE statement:

1. The Format 1 REPLACE statement specifies the text of the source program to be replaced by the corresponding text. Each matched occurrence of pseudo-text-1 in the source program is replaced by the corresponding pseudo-text-2.
2. The Format 2 REPLACE statement discontinues any text replacement currently in effect.

-
3. Each occurrence of the REPLACE statement is in effect from the point at which it is specified until the next occurrence of the statement or until the end of the separate program.
 4. Any REPLACE statements contained in a source program are processed after any COPY statements contained in a source program.
 5. The text produced as a result of processing a REPLACE statement must not contain a REPLACE statement.
 6. The comparison operation to determine text replacement occurs in the following manner:
 - a. Starting with the leftmost source program text word and the first pseudo-text-1, pseudo-text-1 is compared to an equivalent number of contiguous source program text words.
 - b. pseudo-text-1 matches the source program text if the order sequence of text words forming 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, or space in pseudo-text-1 or in the source program text is considered to be a single space. Each sequence of one or more space separators is considered to be a single space.
 - c. If no match occurs, the comparison is repeated with each next successive occurrence of pseudo-text-1, until either a match is found or there is no next successive occurrence of pseudo-text-1.
 - d. 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.
 - e. 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 been matched or been considered as a leftmost source program text word and completed the comparison cycle.
 7. 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. Refer to "Reference Format" on page 3-4. 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. A comment line or blank line in source program text is not replaced if the line appears within the sequence of text words matching pseudo-text-1.
 8. Debugging lines are permitted in pseudo-text. Text words within a debugging line are treated as if the D did not appear in the indicator area.
 9. 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.
 10. 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. Refer to "Reference Format" on page 3-4. When inserting text words of pseudo-text-2 into the source program, additional spaces may be introduced only between text words where there already exists a space (including the assumed space between source lines).

-
11. If additional lines are introduced into the source program as a result of processing REPLACE statements, the indicator area of the introduced lines contains the same character as the first line of the replaced text, unless the line contains a hyphen, in which case the introduced line contains a space.
 12. If any literal within pseudo-text-2 is too long to be accommodated on a single line without continuation, and the literal is not being placed on a debugging line, additional continuation lines are introduced which contain the remainder of the literal. If replacement requires the literal to be continued on a debugging line, the program is in error.

Chapter 10. Listing Control

Contents

About This Chapter	10-3
SKIP1, SKIP2, and SKIP3 Statements	10-4
Function	10-4
General Format	10-4
Syntax Rule	10-4
General Rules	10-4
EJECT Statement	10-5
Function	10-5
General Format	10-5
Syntax Rule	10-5
General Rule	10-5
TITLE Statement	10-6
Function	10-6
General Format	10-6
Syntax Rules	10-6
General Rules	10-7

About This Chapter

This chapter describes five compiler-directed listing control statements. It also describes how the statements improve the readability of a program listing by the vertical spacing, ejection, and page title of the listing produced by the COBOL compiler.

SKIP1, SKIP2, and SKIP3 Statements

Function

The SKIP1, SKIP2, and SKIP3 statements control the vertical spacing of the source code listing produced by the AIX VS COBOL system. They specify the lines to be skipped in the source code listing. OSVS VSC2

General Format

The following figures show the format of the SKIP statement:



Syntax Rule

These statements may begin either in area A or in area B, and must be the only statement on the line. It may be followed by a period. OSVS VSC2

General Rules

The following general rules apply to the SKIP statements: OSVS VSC2

- 1. SKIP1 tells AIX VS COBOL system to skip one line (double spacing).
SKIP2 tells AIX VS COBOL system to skip two lines (triple spacing).
SKIP3 tells AIX VS COBOL system to skip three lines (quadruple spacing).*
- 2. The SKIP statement itself is not printed.*

EJECT Statement

Function

The EJECT statement tells the AIX VS COBOL system to print the next line of source code at the top of the next page. OSVS VSC2

General Format

The following figure shows the format of the EJECT statement: OSVS VSC2

▶— EJECT ▶

Syntax Rule

EJECT may begin either in area A or in area B and must be the only statement on the line. It may be followed by a period. OSVS VSC2

General Rule

The EJECT statement itself is not printed. OSVS VSC2

TITLE Statement

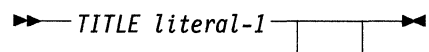
Function

The TITLE statement tells the AIX VS COBOL system what title to print on the first line of all following pages of the program listing. VSC2

General Format

The following figure shows the format of the TITLE statement: VSC2

▶— *TITLE literal-1* —▶



Syntax Rules

The following syntax rules apply to the TITLE statement: VSC2

- literal-1 must be nonnumeric and may be followed by a period.*
- The word TITLE may begin either in area A or area B and must be the only statement on the line.*
- The TITLE statement may appear anywhere in the source program.*

DBCS Support

- literal-1 may be a Double-Byte Character Set (DBCS) literal.*

End of DBCS Support

General Rules

The following general rules apply to the TITLE statement:

VSC2

- 1. literal-1 is used as a title on all following pages of the program listing. The default title, used until a TITLE directive is encountered, identifies the AIX VS COBOL system and its current release level.*
- 2. The chosen or default title occupies the left side of the first line of each page. The remainder of the line gives the date and time the intermediate code is produced and the page number.*
- 3. A second title line is output containing the name of the main source file and of the current COPY file.*
- 4. The TITLE statement causes an immediate new page.*
- 5. The TITLE statement itself is not printed.*



Chapter 11. Interprogram Communication

Contents

About This Chapter	11-5
Introduction	11-6
Language Concepts	11-6
Nested Source Programs	11-6
File Connector	11-6
Global Names and Local Names	11-6
External Objects and Internal Objects	11-7
Common Programs and Initial Programs	11-7
Sharing Data	11-8
Sharing Files	11-8
Scope of Names	11-8
Nested Source Programs	11-10
General Description	11-10
Organization	11-11
Structure	11-11
Initial State of a Program	11-12
END PROGRAM Header	11-13
Function	11-13
General Format	11-13
Syntax Rules	11-13
General Rules	11-13
Example	11-14
Identification Division in the Interprogram Communication Module	11-15
PROGRAM-ID Paragraph and Nested Source Programs	11-15
Example	11-16
Data Division in the Interprogram Communication Module	11-17
LINKAGE SECTION	11-17
Noncontiguous Linkage Storage	11-18
Linkage Records	11-18
Initial Values	11-18
File Description Entry in the Interprogram Communication Module	11-19
Function	11-19
General Format	11-19
Syntax Rules	11-24
General Rules	11-24
Data Description Entry in the Interprogram Communication Module	11-25
Function	11-25
General Format	11-25
Syntax Rules	11-26
General Rule	11-27
Report Description Entry in the Interprogram Communication Module	11-28
Function	11-28
General Format	11-28
Syntax Rule	11-28
General Rules	11-29
EXTERNAL Clause	11-30
Function	11-30
General Format	11-30
Syntax Rules	11-30
General Rules	11-30
GLOBAL Clause	11-31
Function	11-31
General Format	11-31
Syntax Rules	11-31
General Rules	11-31
Example	11-32
Procedure Division in the Interprogram Communication Module	11-33
Procedure Division Header	11-33
Example	11-34

CALL Statement	11-36
Function	11-36
General Format	11-36
Syntax Rules	11-37
General Rules	11-38
Example 1	11-41
Example 2	11-42
CANCEL Statement	11-43
Function	11-43
General Format	11-43
Syntax Rules	11-43
General Rules	11-43
CHAIN Statement	11-45
Function	11-45
General Format	11-45
Syntax Rules	11-46
General Rules	11-46
ENTRY Statement	11-47
Function	11-47
General Format	11-47
Syntax Rules	11-47
General Rules	11-48
Example	11-48
EXIT PROGRAM Statement	11-50
Function	11-50
General Format	11-50
Syntax Rules	11-50
General Rules	11-50
GOBACK Statement	11-51
Function	11-51
General Format	11-51
Syntax Rule	11-51
General Rules	11-51
USE Statement	11-52
Function	11-52
General Format	11-52
Syntax Rule	11-52
General Rules	11-52
USE BEFORE REPORTING Statement	11-53
Function	11-53
General Format	11-53
General Rules	11-53



About This Chapter

This chapter describes the Interprogram Communication module and how it supports the following control and information transfer between programs:

- Transferring control from one program to another
- Passing parameters between calling and called programs
- Sharing data and/or files between several programs.

Introduction

The interprogram communication module provides a facility by which a program can communicate with one or more programs. This provides the programmer with a modular programming capability and the ability to load modules dynamically. Interprogram communication is provided by:

- The ability to transfer control from one program to another within a run-unit
- The ability for both programs to have access to the same data-items
- The ability for multiple programs to share data and files.

Language Concepts

This section describes language concepts regarding interprogram communication.

Nested Source Programs

A COBOL source program may contain other COBOL source programs, and these contained programs may reference some of the resources of the program in which they are contained.

When a program, program B, is contained in another program, program A, it may be directly contained or indirectly contained. Program B is directly contained in program A if there is no program contained in program A that also contains program B. Program B is indirectly contained in program A if a program contained in program A also contains program B.

File Connector

A file connector is a storage area containing information about a file. File connectors are used as the linkage between a file name and a physical file and between a file name and its associated record area.

Global Names and Local Names

A data name names a data-item. A file name names a file connector. These names are classified as either global or local.

A global name may be used to refer to the object either from within the program which declares the global name, or from within any program contained in the program which declares the global name. However, a local name may be used only to refer to the object from within the program which declares the local name.

Some names are always global. Other names are always local. Yet other names are local or global depending upon specifications in the program declaring the names.

A record name is global if the GLOBAL clause is specified in its record description entry. When the record description entries are in the FILE SECTION, the GLOBAL clause is specified in the file description entry for the file name associated with the record description entry.

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

A condition name declared in a data description entry is global if that entry is subordinate to another entry in which the GLOBAL clause is specified. However, specific rules sometimes prohibit specification of the GLOBAL clause for certain data description, file description, or record description entries.

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

If a data name, a file name, or a condition name declared in a data description entry is not global, the name is local.

Global names are transitive across programs contained within other programs.

External Objects and Internal Objects

Accessible data-items usually require that certain representations of data be stored. File connectors usually require that certain information concerning files be stored. The storage associated with a data-item or a file connector may be external or internal to the program in which the object is declared.

A data-item or file connector is external if the object's storage is associated with the run-unit rather than with any particular program within the run-unit. An external object may be referenced by any program in the run-unit which describes the object. References to an external object from different programs using separate descriptions of the objects are always to the same object. In a run-unit, there is only one representative of an external object.

An object is internal if the storage associated with the object is associated only with the program describing the object.

External and internal objects may 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 in which it is described.

A file connector is given the external attribute by the presence of the EXTERNAL clause in the associated file description entry. If the file connector does not have the external attribute, it is internal to the program in which the associated file name is described.

The data record described as subordinate to a file description entry not containing the EXTERNAL clause or a sort-merge file description entry, and any data-items described as subordinate to the data description entries for such records, are always internal to the program 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.

Data records, subordinate data-items, and various associated control information described in the LINKAGE, COMMUNICATION, and REPORT SECTIONS of a program are always considered to be internal to the program describing that data. Special considerations apply to data described in the LINKAGE SECTION, whereby an association is made between the data records described and other data-items accessible to other programs.

Common Programs and Initial Programs

All programs forming part of a run-unit may possess none, one, or both of the following attributes: common and initial.

A common program is one which, despite being directly contained within another program, may be called by any program directly or indirectly contained in that other program. The common attribute is attained by specifying the COMMON clause in a program's Identification Division. The COMMON clause allows subprograms to be used by all the programs contained within a program.

An initial program is one whose program state is initialized when the program is called. Thus, whenever an initial program is called, its program state is the same as when the program was first called in that run-unit. The process of initializing an initial program, initializes the program's internal data. Therefore, an item of the program's internal data whose description contains a VALUE clause is initialized to that defined value, but an item whose description does not contain a VALUE clause is initialized to a value depending on the DEFAULTBYTE directive. Refer to the *User's Guide*. 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. The initial attribute is attained by specifying the INITIAL clause in the program's Identification Division.

Sharing Data

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

- The data content of an external data record may be referenced from any program provided that program has described that data record.
- If a program is contained within another program, both programs may refer to data possessing the global attribute either in the containing program or in any program which directly or indirectly contains the containing program.
- The mechanism whereby a parameter value is passed by reference from a calling program to a called program establishes a common data item. The called program, which may use a different identifier, may refer to a data-item in the calling program.

Sharing Files

Two programs in a run-unit may reference common file connectors in the following circumstances:

1. An external file connector may be referenced from any program that describes that file connector.
2. If a program is contained within another program, both programs may refer to a common file connector by referring to an associated global file name either in the containing program or in any program which directly or indirectly contains the containing program.

Scope of Names

When programs are directly or indirectly contained within other programs, each program may use identical user-defined words to name objects, independent of the use of these user-defined words by other programs. Refer to "User-Defined Words" on page 2-7. When identically named objects exist, a program's reference to such a name, even when it is a different type of user-defined word, is to the object which the program describes rather than to the object possessing the same name, described in another program.

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

- cd-name
- paragraph-name
- section-name

The following types of user-defined words may be referenced by any COBOL program:

- library-name
- text-name

The following types of user-defined words, when they are declared in a COMMUNICATION SECTION, may be referenced only by statements and entries in that program which contains the section:

- condition-name
- data-name
- record-name

The following types of names, when declared within a CONFIGURATION SECTION, may be referenced only by statements and entries either in the program containing the CONFIGURATION SECTION or in any program contained within that program:

- alphabet-name
- class-name
- condition-name
- mnemonic-name
- symbolic-character

Specific conventions, for declarations and references, apply to the following types of user-defined words when the conditions listed above do not apply:

- condition-name
- data-name
- file-name
- index-name
- program-name
- record-name
- report-name

Conventions for Program Names

The name of a program is declared in the PROGRAM-ID paragraph of the Identification Division. A program name may be referenced only by the CALL statement, the CHAIN statement, the CANCEL statement, and the END PROGRAM header. The program names allocated to programs in a run-unit are not necessarily unique but, when two of the programs are identically named, at least one of those programs must be directly or indirectly contained within another program which does not contain the other of those two programs.

The following rules regulate the scope of a program name:

1. If the program does not possess the COMMON attribute and is directly contained within another program, the program's name may be referenced only by statements included in the containing program.
2. If the program does possess the COMMON attribute and is directly contained within another program, the program's name may be referenced only by statements included in the containing program and any programs directly or indirectly contained, except programs possessing the COMMON attribute and any programs contained within it.
3. If the named program is separately compiled, the program's name may be referenced by statements included in any other program in the run-unit, except programs it directly or indirectly contains.

Conventions for Condition Names, Data Names, File Names, Record Names and Report Names

When condition names, data names, file names, record names, and report names are declared in a source program, these names may be referenced only by that program, except when one or more of the names is global and the program contains other programs.

The requirements governing the uniqueness of the names allocated by a single program to be condition names, data names, file names, record names, and report names are explained in "User-Defined Words" on page 2-7.

A program cannot reference any condition name, data name, file name, record name, or report name declared in any program it contains.

A global name may be referenced in the program in which it is declared or in any programs directly or indirectly contained within that program.

When a program, program B, is directly contained within another program, program A, both programs may define a condition name, a data name, a file name, a record name, or a report name using the same user-defined word. When such a duplicate name is referenced in program B, the following rules are used to determine the referenced object:

1. The set of names to be used for determination of a referenced object consists of all names defined in program B and all global names that are defined in program A, and in any programs that 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 objects is identified.
2. If only one object is identified, it is the referenced object.
3. If more than one object is identified, no more than one of them can have a name local to program B. If zero or one of the objects has a name local to program B, the following rules apply:
 - a. If the name is declared in program B, the object in program B is the referenced object.
 - b. Otherwise, if program A is contained within another program, the referenced object is:
 - 1) The object in program A, if the name is declared in program A.
 - 2) The object in the containing program, if the name is not declared in program A and is declared in the program containing program A. This rule is applied to further containing programs until a single valid name is found.

Conventions for Index Names

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 the scope of the data name specifying the table with the named index. The scope of the name rules for data names also apply to index names. Index names cannot be qualified.

If a data-item possessing the EXTERNAL attribute includes a table accessed with an index, that index does not automatically possess the EXTERNAL attribute.

Nested Source Programs

This section describes the organization, structure, format, syntax and general rules of nested source programs.

General Description

A COBOL source program may contain other COBOL source programs. The contained programs may reference some of the resources of the programs in which they are contained.

Organization

With the exception of COPY and REPLACE statements and the END PROGRAM header, the statements, entries, paragraphs, and sections of a COBOL source program are grouped into four divisions which are sequenced in the following order:

1. The Identification Division
2. The Environment Division
3. The Data Division
4. The Procedure Division

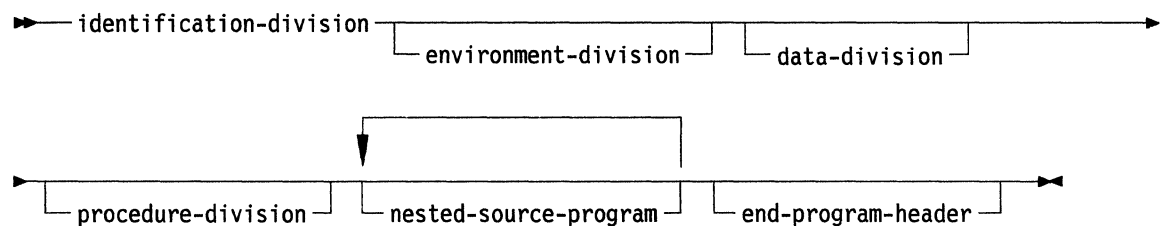
The end of a COBOL source program is indicated by either the END PROGRAM header or by the absence of additional source program lines.

Structure

This section describes the general format and order of presentation of the entries and statements which constitute a COBOL source program. The generic terms identification division, environment division, data division, procedure division, nested source program, and end program header represent a COBOL Identification Division, a COBOL Environment Division, a COBOL Data Division, a nested COBOL Procedure Division, a nested COBOL source program, and a COBOL END PROGRAM header, respectively.

General Format

The following figure shows the general format of a COBOL source program:



Syntax Rules

END PROGRAM header must be present if:

1. The COBOL source program contains one or more nested COBOL source programs, or
2. The COBOL source program is contained within another COBOL source program.

General Rules

The following general rules apply:

1. The beginning of a division in a program is indicated by the appropriate division header. The end of a division is indicated by one of the following:
 - a. The next division header in the program
 - b. An Identification Division header indicating the start of another source program
 - c. The END PROGRAM header
 - d. The physical position after which no more source program lines occur.
2. A COBOL source program directly or indirectly contained in another program may reference certain resources defined in the containing program.

-
3. The object code produced from passing a source program contained in another program through the AIX VS COBOL compiler is inseparable from the object code produced from the containing program.

Initial State of a Program

The initial state of a program is the state of a program the first time it is called in a run-unit.

Characteristics of a Program

The following are characteristics of a program:

1. The internal data contained in the WORKING-STORAGE SECTION and the COMMUNICATION SECTION of the program are initialized. If a VALUE clause is used in the description of the data-item, the data item is initialized to the specified value. If a VALUE clause is not associated with a data-item, the initial value of the data-item is undefined.
2. Files with internal file connectors associated with the program are not in the open mode.
3. The control mechanisms for all PERFORM statements contained in the program are set to their initial states.
4. Any GO TO statement referred to by an ALTER statement contained in the same program is set to its initial state.

Programs in the Initial State

A program is in the initial state:

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

END PROGRAM Header

Function

The END PROGRAM header indicates the end of the named COBOL source program.

General Format

The following figure shows the format of the END PROGRAM header:

▶▶— END PROGRAM program-name —▶▶

Syntax Rules

The following syntax rules apply for the END PROGRAM header:

1. The program name must conform to the rules for forming a user-defined word.
2. The program name must be identical to a program name declared in a preceding PROGRAM-ID paragraph. Refer to “PROGRAM-ID Paragraph and Nested Source Programs” on page 11-15.
3. If a PROGRAM-ID paragraph declaring a specific program name is stated between the PROGRAM-ID paragraph and the END PROGRAM header for another program name, the END PROGRAM header for the former program must precede the END PROGRAM header referencing the latter program name.

General Rules

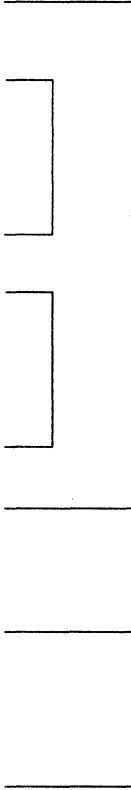
The following rules apply to the END PROGRAM header:

1. The END PROGRAM header must be present in every program that contains or is contained in another program.
2. The END PROGRAM header indicates the end of the specified COBOL source program.
3. When the program terminated by the END PROGRAM header is contained in another program, the next statement must either be an Identification Division header or another END PROGRAM header which terminates the containing program.
4. If the program terminated by the END PROGRAM header is not a contained program, the next COBOL statement, if any, must be the Identification Division header of the next program in the source file.

Example

The following example shows nested COBOL programs:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. PAYROLL.  
  
IDENTIFICATION DIVISION.  
PROGRAM-ID. POSTING.  
:  
:  
:  
END PROGRAM POSTING.  
  
IDENTIFICATION DIVISION.  
PROGRAM-ID. WEEKLY.  
:  
:  
:  
END PROGRAM WEEKLY.  
  
END PROGRAM PAYROLL.  
:  
:  
:  
IDENTIFICATION DIVISION.  
PROGRAM-ID. INVENTORY.  
:  
:  
:  
END PROGRAM INVENTORY.
```



In the above example, the program PAYROLL contains the nested programs POSTING and WEEKLY. The END PROGRAM headers referring to POSTING and WEEKLY must precede the header referring to PAYROLL. INVENTORY is in the same source file but is not contained in another program.

Identification Division in the Interprogram Communication Module

This section describes the Identification Division in the interprogram communication module.

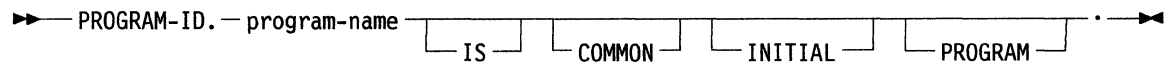
PROGRAM-ID Paragraph and Nested Source Programs

Function

The PROGRAM-ID paragraph specifies the name by which a program is identified and assigns selected program attributes to that program.

General Format

The following figure shows the format of the PROGRAM ID paragraph:



Syntax Rules

The following syntax rules apply to the PROGRAM-ID paragraph:

1. The program name must conform to the rules for formation of a user-defined word.
2. A program contained within another program may not be assigned the same name as any other program in the containing program.
3. The optional COMMON clause may be used only if the program is contained within another program.
4. If the IS PROGRAM phrase is present, either COMMON or INITIAL or both must be specified. When both are specified, the order is irrelevant.

General Rules

The following rules apply to the PROGRAM-ID paragraph:

1. The COMMON clause specifies that the program is a common program. A common program is contained in another program but may be called from programs other than the containing program.
2. The INITIAL clause specifies that the program is an initial program. When an initial program is called, it and any programs contained in it are returned to their initial state. Refer to "Initial State of a Program" on page 11-12.

Example

The following program skeleton shows program structure and legal and illegal calls:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. A.  
:  
:  
PROCEDURE DIVISION.  
:  
:  
CALL "B" USING ... . (legal)  
:  
:  
CALL "C" USING ... . (legal)  
:  
:  
  
IDENTIFICATION DIVISION.  
PROGRAM-ID. B.  
:  
:  
END PROGRAM B.  
  
IDENTIFICATION DIVISION.  
PROGRAM-ID. C IS COMMON PROGRAM.  
:  
:  
END PROGRAM C.  
  
IDENTIFICATION DIVISION.  
PROGRAM-ID. D.  
:  
:  
PROCEDURE DIVISION.  
:  
:  
CALL "B" USING ... . (illegal, B is not COMMON)  
:  
:  
CALL "C" USING ... . (legal)  
:  
:  
END PROGRAM D.  
END PROGRAM A.
```

Data Division in the Interprogram Communication Module

This section describes Data Division in the interprogram communication module.

LINKAGE SECTION

Except as described in "SET Statement" on page 7-84, the LINKAGE SECTION in a program is meaningful only if the object program functions under the control of a CALL statement containing a USING phrase. VSC2

The LINKAGE SECTION is used in the invoked program for describing data available through the invoking program, but the LINKAGE SECTION is referred to in both the invoking and the invoked program. *(Such data also may be described in the FILE and WORKING-STORAGE SECTIONS of the program.)* MF

No space is allocated in the program for data items referenced by data-names in the LINKAGE SECTION. Procedure Division references to these data items are resolved at run time by equating the reference in the invoked program to the location used in the invoking program. In the case of index names, no such correspondence is established. Index names in the invoked and invoking program always refer to separate indices.

Data items defined in the LINKAGE SECTION of the invoked program may be referenced with the Procedure Division of the invoked program only if both of these statements are true:

1. The data items are specified as operands of the USING phrase of the Procedure Division header or are subordinate to such operands.
2. The object program is under the control of a CALL statement specifying a USING phrase.

The structure of the LINKAGE SECTION is the same as the structure for the WORKING-STORAGE SECTION. The LINKAGE SECTION begins with a section header, followed by data description entries for noncontiguous data items, and/or record description entries.

Each LINKAGE SECTION record name and noncontiguous item name must be unique in the invoked program since it cannot be qualified. Of those items defined in the LINKAGE SECTION, only the following data items may be referenced in the Procedure Division:

- Data-names in the USING phrase of the Procedure Division header
- Data items subordinate to the data-names
- Condition names and/or index names associated with the data-names and/or subordinate data items.

An ADDRESS special register is maintained for each record (01 or 77 level item) in the LINKAGE and WORKING-STORAGE SECTION. These special registers can be specified in the USING phrase, allowing the address of a record to be passed or received. VSC2 MF

Noncontiguous Linkage Storage

Items in the LINKAGE SECTION with no hierarchic relationship to one another need not be grouped into records. They are defined as noncontiguous elementary items. Each of the data items is defined in a separate data description entry beginning with the special level number 77.

The following data clauses are required in each data description entry:

- Level-number 77
- Data-name
- The PICTURE clause or the USAGE IS INDEX clause *or the USAGE IS POINTER clause.* VSC2

Other data description clauses are optional and can be used to complete the description of the item if necessary.

Linkage Records

Data elements in the LINKAGE SECTION with a definite hierarchic relationship to one another must be grouped into records according to the rules for forming record descriptions. Any clause used in an input or output record description can be used in a LINKAGE SECTION.

Initial Values

The VALUE clause must not be specified in the LINKAGE SECTION except in condition name entries (level 88).

The VALUE clause is allowed, and is documentary. OSVS VSC2

File Description Entry in the Interprogram Communication Module

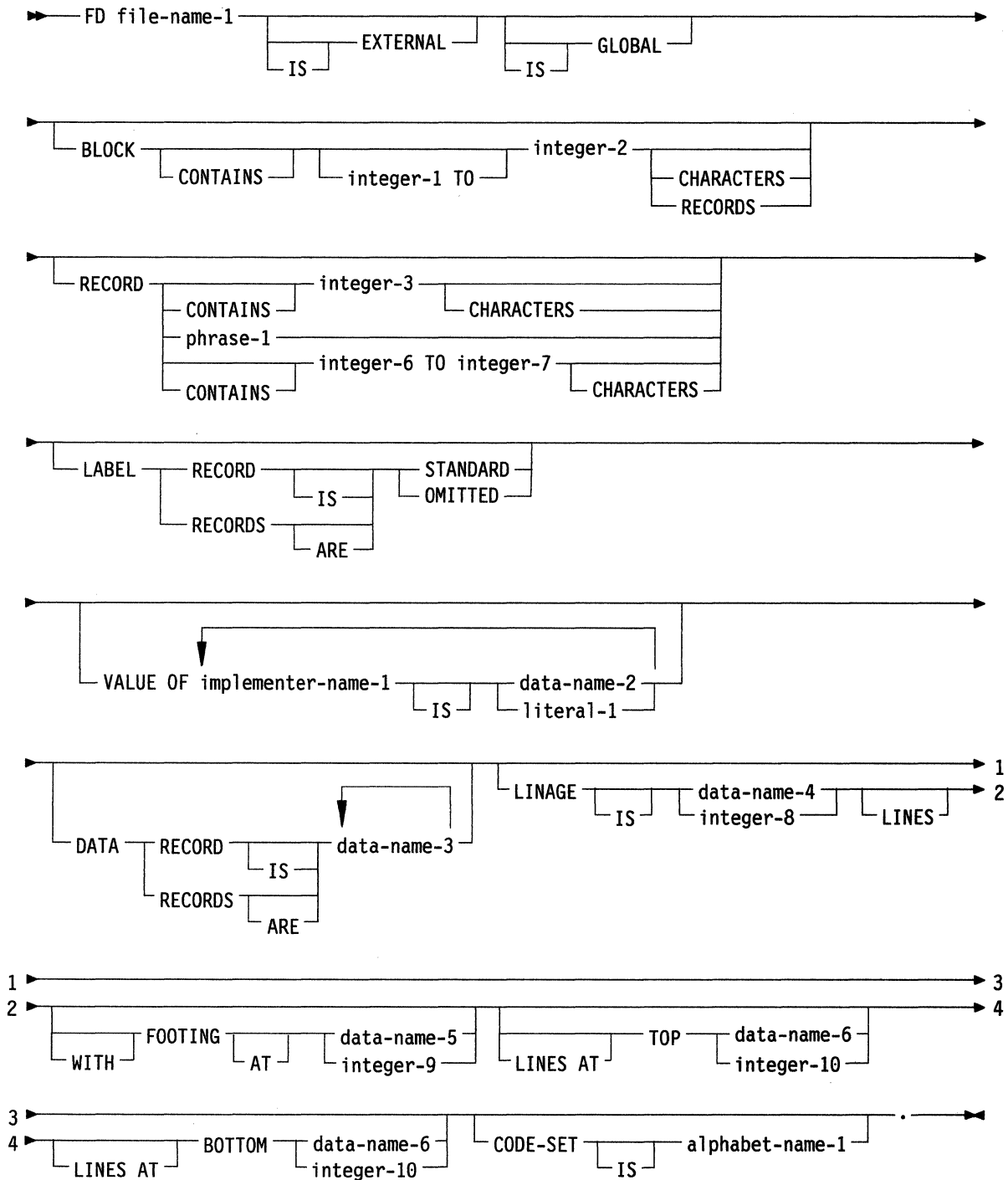
Function

In the Interprogram Communication module, the file description entry in the **FILE SECTION** determines the internal or external attributes of a file connector, of the associated data records, and of the associated data items. The file description entry also determines whether a file name is a local name or a global name.

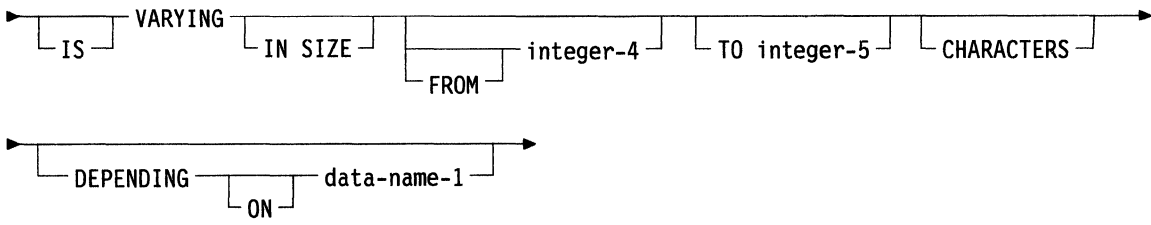
General Format

The following figures show the field description entry formats:

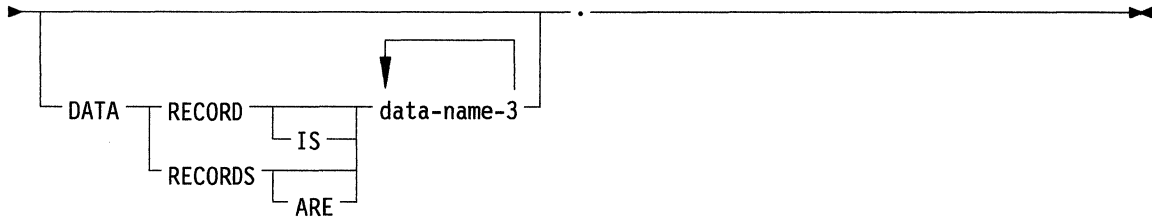
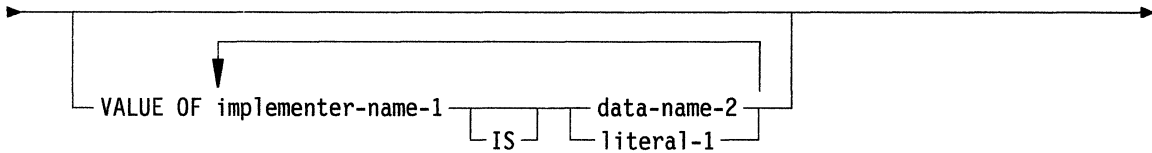
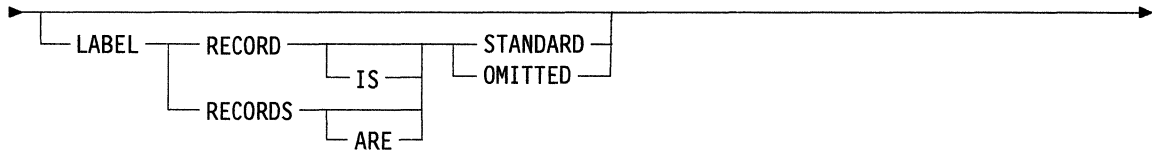
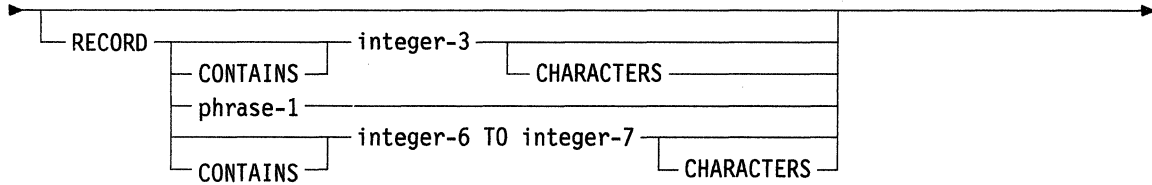
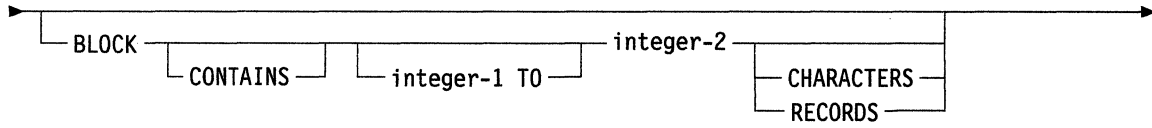
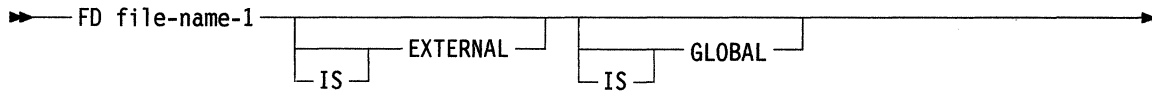
Format 1



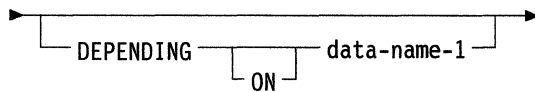
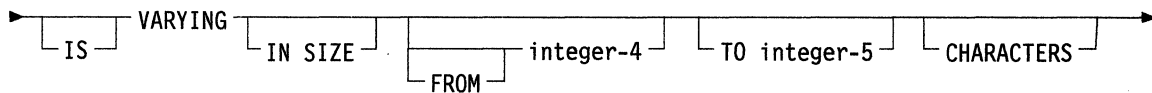
where phrase-1 is:



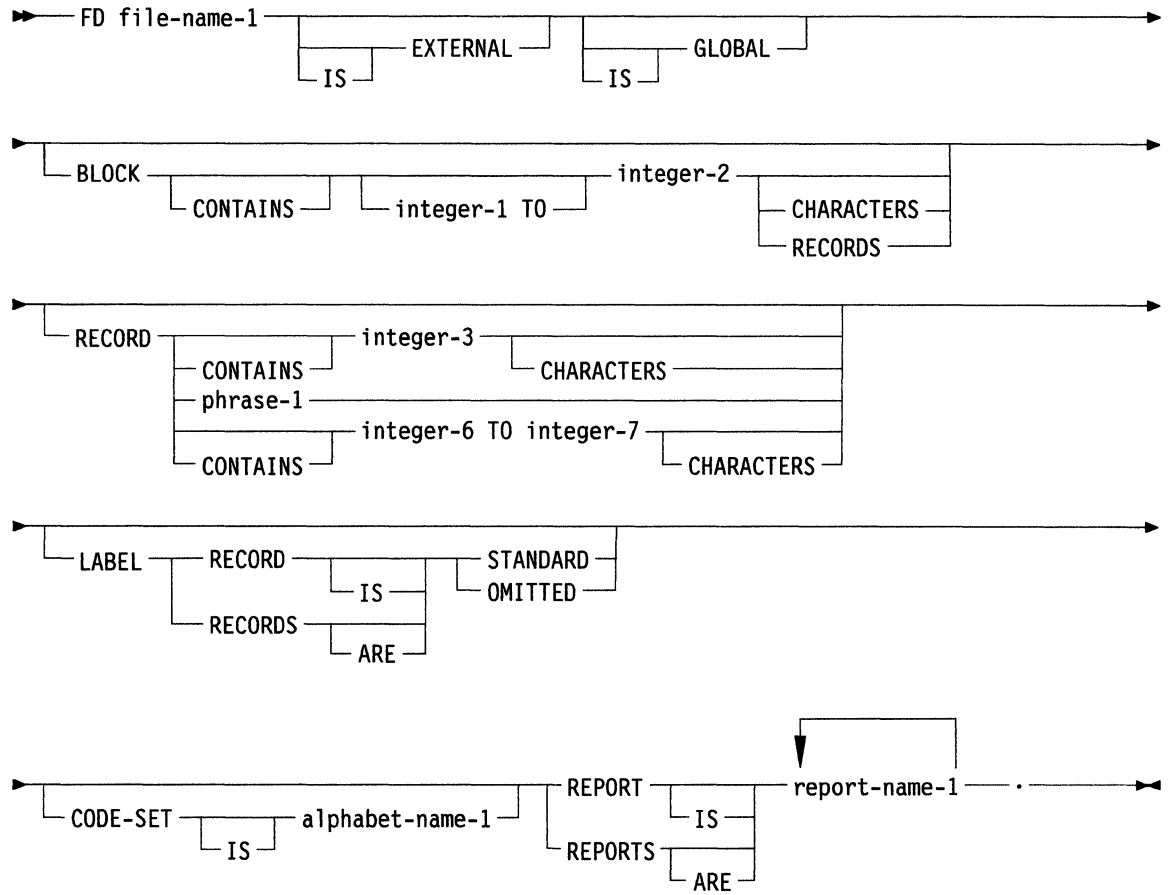
Format 2



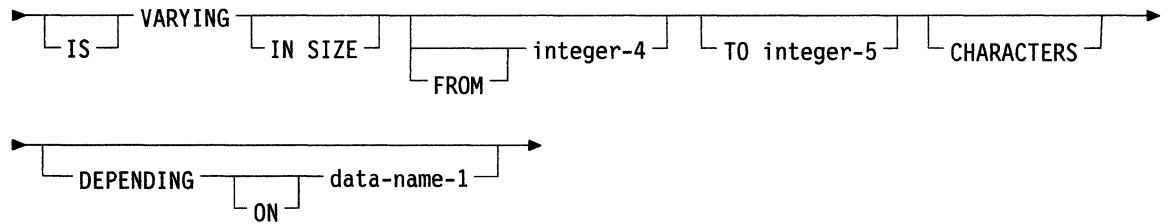
where phrase-1 is:



Format 3



where phrase-1 is:



Syntax Rules

The following syntax rules apply to the file description entry:

1. Format 1 is the file description entry for a sequential file. Refer to “I-O Status” on page 8-8.
2. Format 2 is the file description entry for a relative file or an indexed file. Refer to “I-O Status” on page 8-8.
3. Format 3 is the file description entry for a report file. Refer to Chapter 14, “Report Writer.”

General Rules

The following rules apply to the file description entry:

1. If the file description entry for a sequential file contains the **LINAGE** and **EXTERNAL** clauses, the **LINAGE-COUNTER** data item is an external data item. If the file description entry for a sequential file contains the **LINAGE** and **GLOBAL** clauses, the special register **LINAGE-COUNTER** is a global name.
2. All other clauses in the file description entry are presented in the appropriate module within these specifications.

Data Description Entry in the Interprogram Communication Module

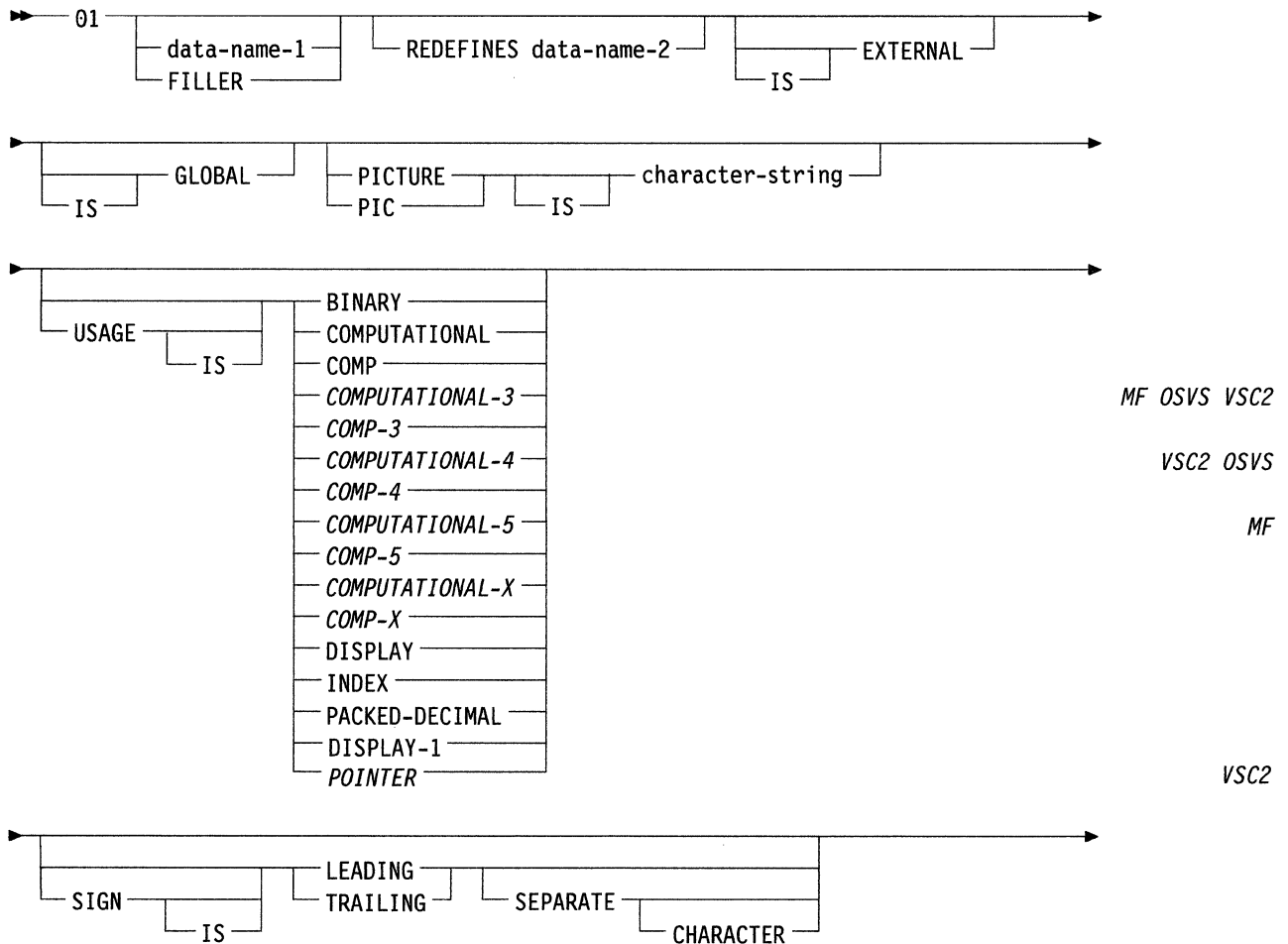
Function

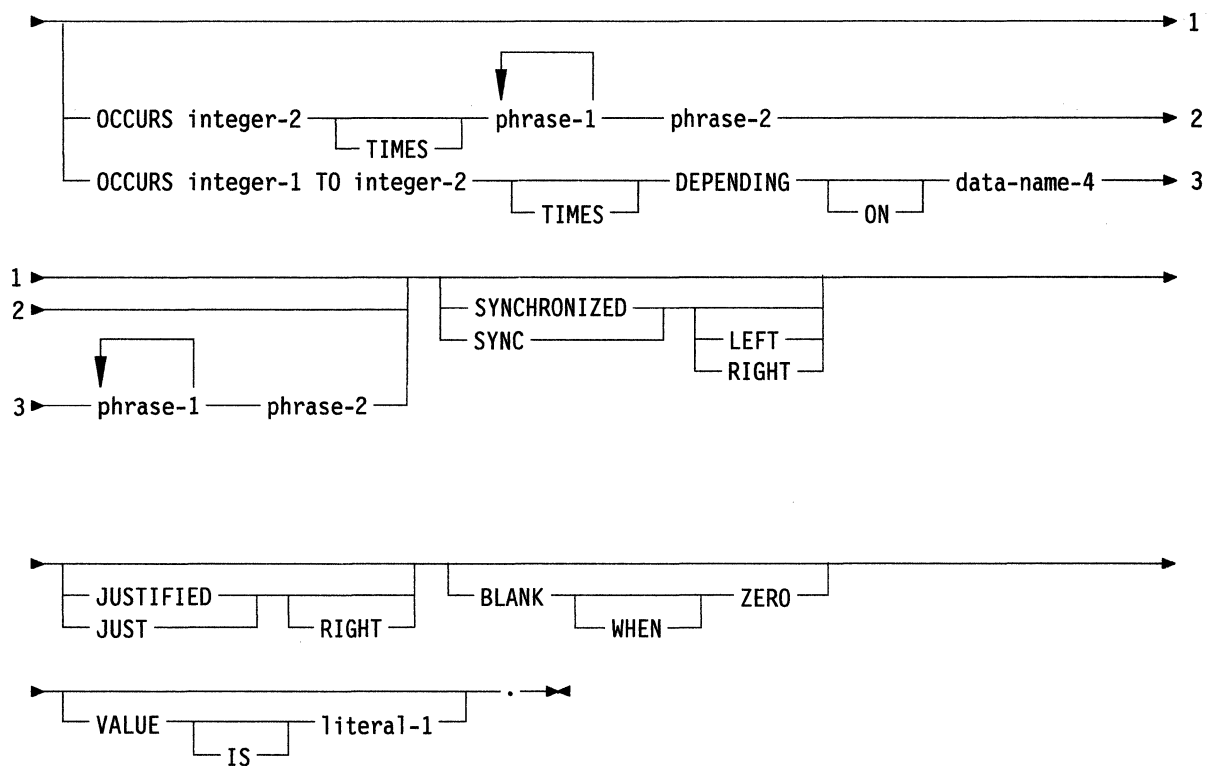
In the Interprogram Communication module, a level 01 data description entry in the WORKING-STORAGE SECTION or FILE SECTION determines whether the data record and its subordinate data items have local names or global names.

In the Interprogram Communication module, a level 01 data description entry in the WORKING-STORAGE SECTION determines the internal or external attribute of the data record and its subordinate data items.

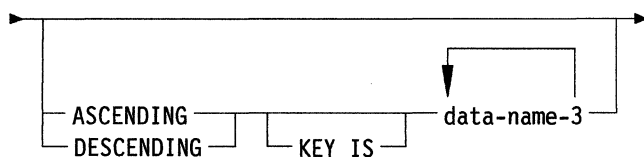
General Format

The following figure shows the format of the data description entry:

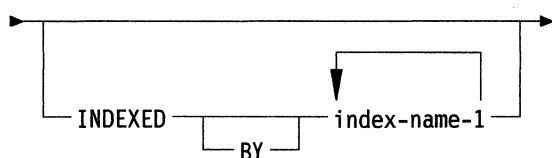




where phrase-1 is:



phrase-2 is:



Syntax Rules

The following syntax rules apply to the data description entry:

1. The EXTERNAL clause may be specified only in level 01 data description entries in the WORKING-STORAGE SECTION.
2. The EXTERNAL clause and the REDEFINES clause must not be specified in the same data description entry.
3. The GLOBAL clause may be specified only in level 01 data description entries.
4. Data-name-1 must be specified for any entry containing the GLOBAL or EXTERNAL clause, or for record description associated with a file description entry containing the EXTERNAL or GLOBAL clause.

General Rule

All other clauses in the data description entry are presented in Chapter 3, "Introduction to the Nucleus" on page 3-1 and Chapter 4, "Identification Division in the Nucleus" on page 4-1.

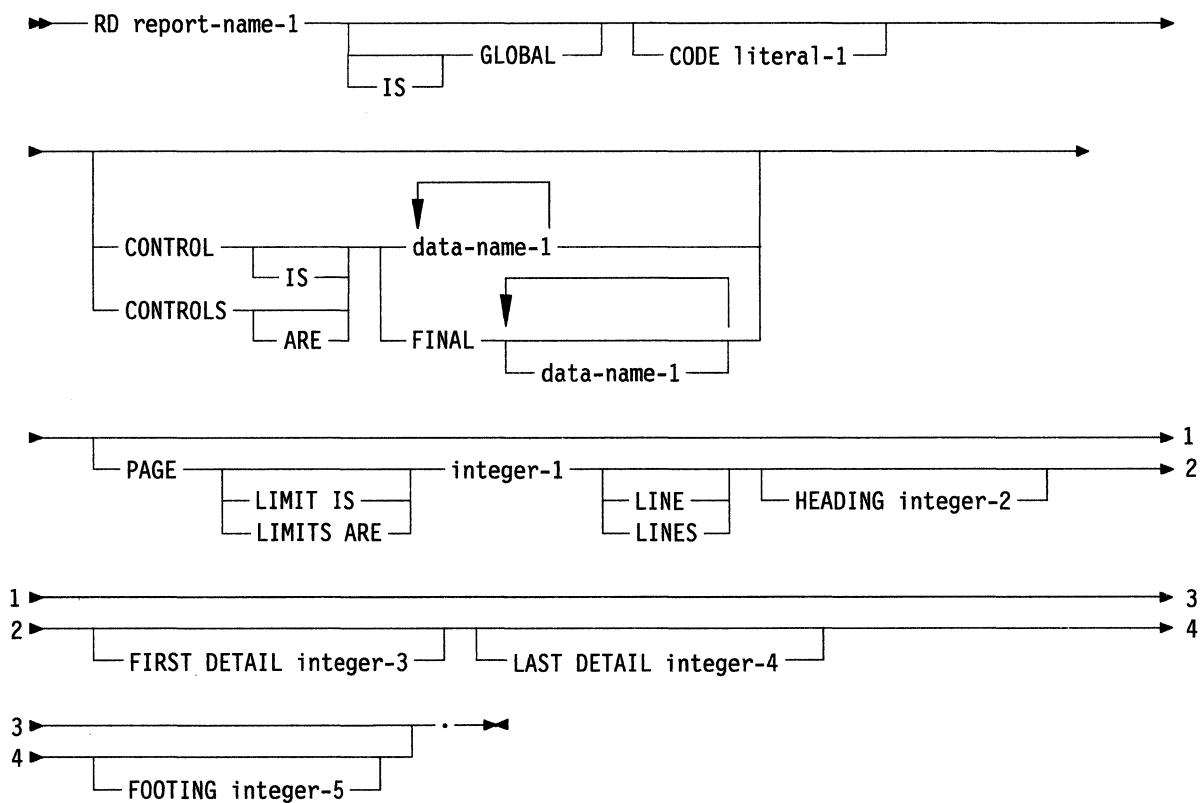
Report Description Entry in the Interprogram Communication Module

Function

Within the Interprogram Communication module, the report description entry in the REPORT SECTION determines whether a report name is a local name or a global name.

General Format

The following figure shows the report description entry:



Syntax Rule

Refer to Chapter 14, "Report Writer."

General Rules

The following general rules apply to the report description entry:

1. If the report description entry contains the GLOBAL clause, the special registers LINE-COUNTER and PAGE-COUNTER are global names.
2. All other clauses in the report description entry are presented in Chapter 14, "Report Writer."

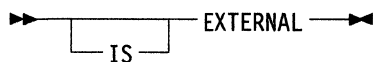
EXTERNAL Clause

Function

The **EXTERNAL** clause specifies that a data item or a file connector is external. The constituent data items and group items of an external data record are available to every program in the run-unit which describes the record.

General Format

The following figure shows the general format **EXTERNAL** clause:



Syntax Rules

The following syntax rules apply to the **EXTERNAL** clause:

1. The **EXTERNAL** clause may be specified in file description entries or in record description entries in the **WORKING-STORAGE SECTION**.
2. The data-name specified as the subject of the level 01 entry with the **EXTERNAL** clause must not be the same data-name specified for any other data description entry, with the **EXTERNAL** clause, in the same program.
3. The **VALUE** clause must not be used in any data description entry that includes, or is subordinate to, an entry including the **EXTERNAL** clause. The **VALUE** clause may be specified for condition name entries associated with such data description entries.

General Rules

The data in the record named by data-name is external and may be accessed and processed by any program in the run-unit which describes or redefines it, subject to the following general rules:

1. If two or more programs describe the same external data record within a run-unit, the record names must be the same and the records must define the same number of standard data format characters. However, a program describing an external record may contain a data description entry with the **REDEFINES** clause, redefining the complete external record, and this complete redefinition need not occur identically in other programs in the run-unit. Refer to “**REDEFINES Clause**” on page 6-29.
2. Use of the **EXTERNAL** clause does not imply that the associated file name or data-name is a global name. Refer to “**GLOBAL Clause**” on page 11-31.
3. The file connector associated with this description entry is an external file connector.
4. Refer to the *User's Guide* for further information on files with the **EXTERNAL** attribute.

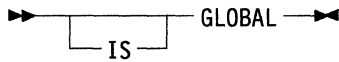
GLOBAL Clause

Function

The GLOBAL clause specifies that a data-name, a file name, or a report name is a global name. A global name is available to every program contained within the program which declares it.

General Format

The following figure shows the GLOBAL clause:



Syntax Rules

The following syntax rules apply to the GLOBAL clause:

1. The GLOBAL clause may be specified only in level 01 data description entries in the FILE or the WORKING-STORAGE SECTION, file description entries, or report description entries.
2. In the same Data Division, the data description entries for any two data items with the same data-name must not include the GLOBAL clause.
3. 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.

General Rules

The following rules apply to the GLOBAL clause:

1. A data-name, file name, or report name described using a GLOBAL clause is a global name. All data-names subordinate to a global name are global names. All condition names associated with a global name are global names.
2. A statement in a program contained directly or indirectly within a program that describes a global name may reference that name without describing it again. Refer to "Scope of Names" on page 11-8.
3. If the GLOBAL clause is used in a data description entry containing the REDEFINES clause, only the subject of the REDEFINES clause possesses the global attribute.

Example

The following example shows nested programs and scopes:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. PAYROLL.  
:  
:  
DATA DIVISION.  
:  
:  
01 TAX-RATES IS GLOBAL.  
05 RATE-85          PIC 99V999.  
05 RATE-86          PIC 99V999.  
05 RATE-87          PIC 99V999.  
05 RATE-88          PIC 99V999.  
01 TOTAL-TAX        PIC 9(6)V99.  
:  
:  
  
IDENTIFICATION DIVISION.  
PROGRAM-ID. WEEKLY.  
:  
:  
ADD 1.01 TO RATE-88.  
ADD 100 TO TOTAL-TAX.  
:  
:  
:  
END PROGRAM WEEKLY.  
  
END PROGRAM PAYROLL.
```

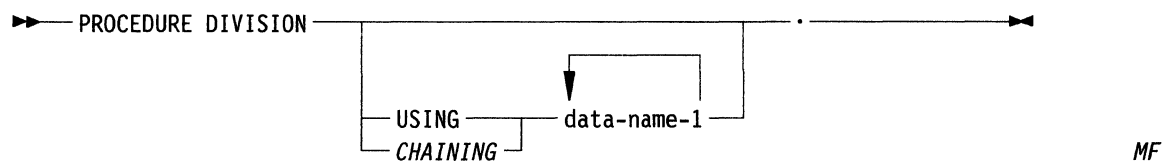
In the example above, program `PAYROLL` contains program `WEEKLY`. The first `ADD` statement in `WEEKLY` is valid because it referred to a globally-defined data item in the containing program. The second `ADD` statement is invalid because `TOTAL-MAX` is not globally defined.

Procedure Division in the Interprogram Communication Module

This section describes the Procedure Division in the interprogram communication module.

Procedure Division Header

The Procedure Division is identified by and must begin with the header:



The USING phrase is present if:

- The object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.
 - *The object program is to receive data resulting from a CHAIN statement, and the CHAIN statement in the chaining program contains a USING phrase.*
- MF*

CHAINING and USING are synonymous.

Each of the data items in the USING phrase of the Procedure Division header may be defined as a level 01 or 77 data item in the LINKAGE SECTION, FILE SECTION or WORKING-STORAGE SECTION of the program in which this header occurs.

MF

An item in the USING phrase can be the name of a data item with USAGE IS POINTER, provided the corresponding item in the USING phrase of the CALL statement is also either the name of a pointer data item or an ADDRESS special register.

VSC2

The LINKAGE SECTION data items are processed according to their data descriptions in the invoked program.

When the USING phrase is present, a match is made between data-name-1 of the Procedure Division header in the invoked program and data-name-1 in the USING phrase of the CALL or CHAIN statement in the invoking program. Their descriptions must define an equal number of character positions. However, they need not be the same name. A data-name must not appear more than once in the USING phrase of the Procedure Division header of the invoked program. However, a data-name may appear more than once in the same USING phrase of a CALL or CHAIN statement.

MF

MF

If data-name is described in the USING phrase of the Procedure Division header in the LINKAGE SECTION, the CALL or CHAIN operation sets the data-name to refer to the corresponding data item in the invoking program. Then, any operation on the data-name in the invoked program operates on the data item in the invoking program.

MF

If any data-name in the USING phrase of the Procedure Division header is described in the FILE SECTION or WORKING-STORAGE SECTION, the CALL or CHAIN operation causes the value of the corresponding data item in the invoking program to be moved to the data item it references. Changes made to the data in the invoked program do not affect the data in the invoking program.

The number of operands in the Procedure Division USING phrase and the CALL USING or CHAIN USING phrase may be different. If they are different, then the operands in the two statements are matched from left to right until the shorter list of operands is exhausted.

If the remaining unmatched operands are in the CALL or CHAIN statement, they are ignored. If the remaining unmatched operands are in the Procedure Division header and are LINKAGE SECTION items, they are unavailable to the program. If they are referenced at run time, a run-time error occurs.

Any mismatch of operands is detected only at run time, so the directive FLAG does not cause this extension to be flagged.

If the USING phrase is specified, the INITIAL clause must not be present in any CD entry. Refer to "Communication Description — Complete Entry Skeleton" on page 15-4.

DBCS Support

Parameters in the Procedure Division USING phrase which correspond to USAGE DISPLAY-1 items in the CALL argument list must be Double-Byte Character Set (DBCS) items.

End of DBCS Support

Example

The following example shows USING and CALL USING phrases:

Calling Program Description

```
WORKING-STORAGE SECTION.
01  PARAM-LIST.
    05  PARTCODE  PIC A.
    05  PARTNO    PIC X(4).
    05  U-SALES   PIC 9(5).
      :
      :
PROCEDURE DIVISION.
      :
      :
      CALL CALLED-PROG
          USING PARAM-LIST.
```

In the calling program, the code for parts (PART CODE) and the part number (PARTNO) are referred to separately.

Called Program Description

LINKAGE SECTION.

01 USING-LIST.

10 PART-ID PIC X(5).

10 SALES PIC 9(5).

:

:

PROCEDURE DIVISION USING USING-LIST.

In the called program, the code for parts and the part number are combined into one data item (PART-ID). Reference to PART-ID is the only valid reference to them.

CALL Statement

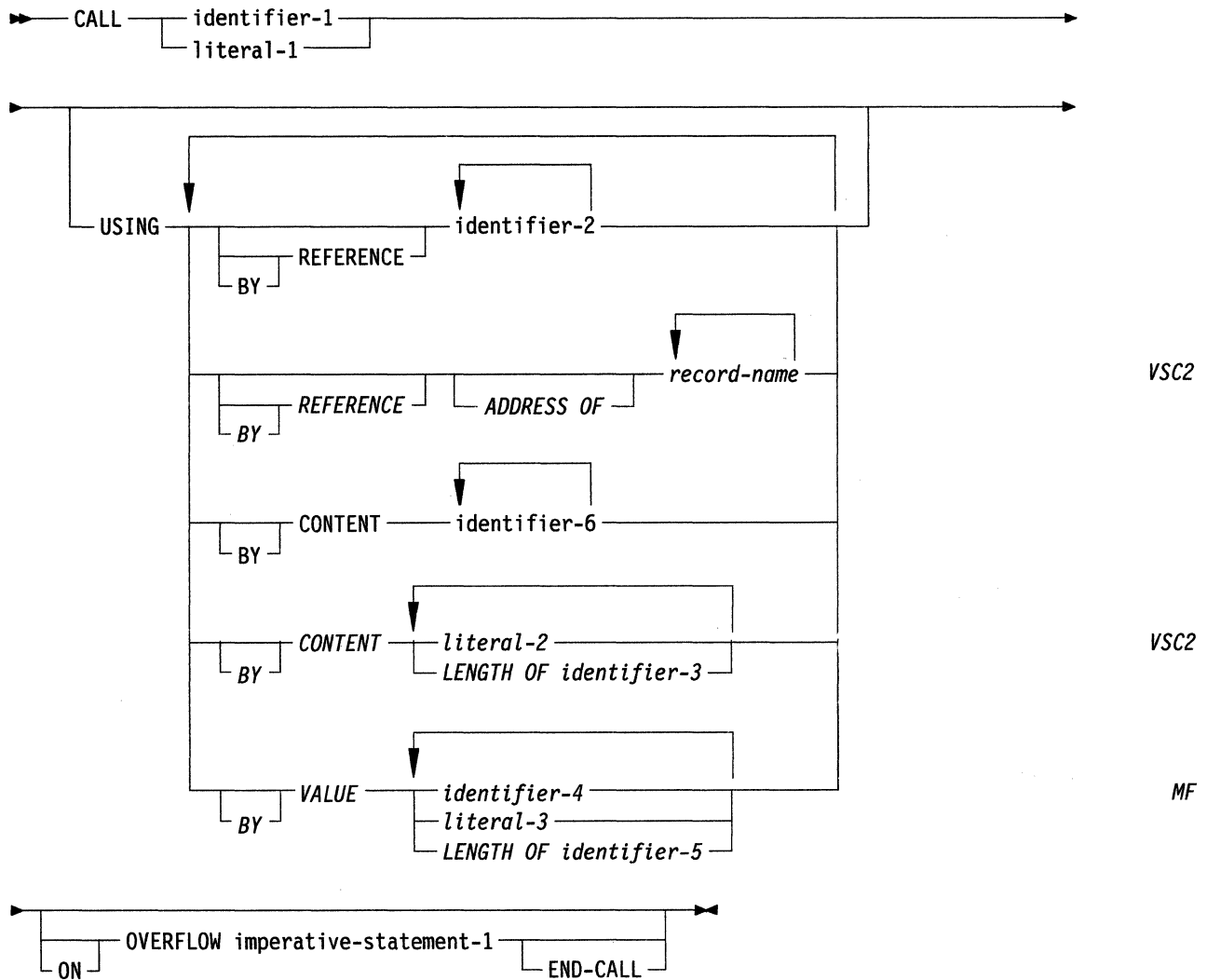
Function

The CALL statement transfers control from one object program to another in the run unit.

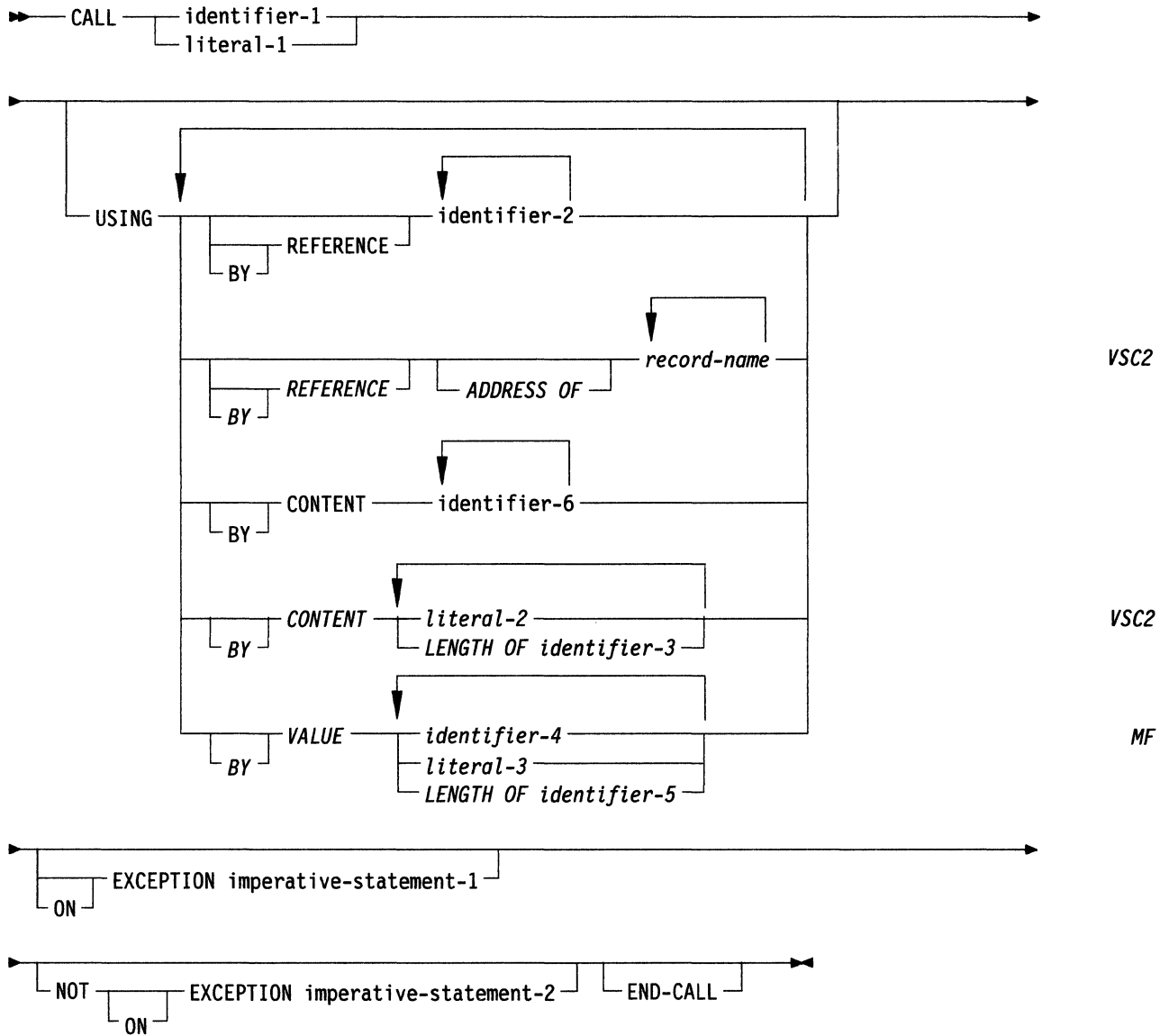
General Format

The following figure shows the formats of the CALL statement:

Format 1



Format 2



Syntax Rules

The following syntax rules apply to the CALL statement:

1. identifier-1 must be defined as an alphanumeric data item. The length of this item cannot exceed 248 bytes. If it is longer, unpredictable results may occur at run time.
2. literal-1 must be a nonnumeric literal.
3. The CALL statement includes a USING phrase only when the Procedure Division header of the called program contains a USING phrase. The two USING phrases should contain the same number of operands. *This rule is not enforced. Refer to "Procedure Division in the Interprogram Communication Module" on page 11-33.*

MF

-
4. Each operand in the USING phrase must be defined as a data item in the FILE SECTION, WORKING-STORAGE SECTION, COMMUNICATION SECTION, or LINKAGE SECTION, and must have a level number of 01 or 77, or 02 through 50. MF
 5. Identifier-1, identifier-2, ... may be qualified when they reference data items defined in the FILE SECTION or the COMMUNICATION SECTION.
 6. *literal-2 must be nonnumeric and cannot be a figurative constant.* VSC2
 7. Record name must name the 01 or 77 level item in the LINKAGE or WORKING-STORAGE SECTION. VSC2 MF
 8. For external calls, literal-1 must match the program-id of the program, including its case (uppercase or lowercase).
 9. Calls from within nested programs to lowercase program names are folded to uppercase.

DBCS Support

10. The arguments in the CALL statement USING phrase should be USAGE DISPLAY-1 items, if the parameters in the Procedure Division USING clause of the called subprogram are DBCS items.

End of DBCS Support

General Rules

The following rules apply to the CALL statement:

1. The execution of a CALL statement passes control to the called program.
2. If the called program has been statically bound into the Run Time Environment, the identified program is known as a called-run-time sub-routine. Such programs retain all the values and settings from the previous call. *If literal-1 or the data referenced by identifier-1 consists entirely of numeric digits or a single non-ASCII character, then it must be statically bound into the Run Time Environment.* MF
3. If the called program has been dynamically loaded, the identified program is a COBOL subprogram. Such a program is loaded from the fixed-disk the first time a run unit calls it and the first time it is called after a CANCEL.

On all other entries into the called program, the state of the program remains unchanged from its state when last exited. This includes the state of all data fields, the status and positioning of all files, and the state of all alterable switch settings.

4. At the end of execution, the called program ignores any ON OVERFLOW or ON EXCEPTION phrase and returns control to the end of the CALL statement or, when the NOT ON EXCEPTION phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement causing explicit transfer of control is executed, control is transferred in accordance with the rules for that statement. Otherwise, control is transferred to the end of the CALL statement upon completion of the execution of imperative-statement-2.
5. If the available run-time memory cannot accommodate the program specified in the CALL statement, the next sequential instruction is executed. If ON OVERFLOW or ON EXCEPTION has been specified, the associated imperative statement is executed and the NOT ON EXCEPTION phrase is ignored.

-
6. The data-names specified by the USING phrase of the CALL statement indicate those items in the calling program that may be referred to in the called program. The data-names must appear in the same order in the USING phrase of the CALL statement and the USING phrase of the Procedure Division header. The corresponding data-names refer to a single set of data. The correspondence is positional, not by name. In the case of index names, no correspondence is established. Index names in the called and calling program always refer to separate indices.
 7. Two or more programs in the run unit may have the same program name, and the reference in a CALL statement to such a program name is resolved by using the scope of names conventions for program names. Refer to "Conventions for Program Names" on page 11-9.

For example, when two programs in the run unit have the same name specified in a CALL statement:

- a. One of the programs must also be contained directly or indirectly in the program including the CALL statement or in the program containing the program including that CALL statement, and
- b. The other program must be contained in the program calling it or in a program containing the program that calls it.

The mechanism used in this example is as follows:

- a. If one of the two programs with the name specified in the CALL statement is directly contained in the program including the CALL statement, that program is called.
 - b. If one of the two programs with the name specified in the CALL statement possesses the COMMON attribute and is directly contained in a program containing another program that includes the CALL statement, that common program is called, unless the calling program is contained in that common program.
 - c. Otherwise, the separate program is called.
8. If the called program does not possess the INITIAL attribute it, and each program contained in it, is in its initial state the first time it is called in a run unit and the first time it is called after a CANCEL.

On all other entries into the called program, the state of the program and each program it contains remains unchanged from its state when last exited.

9. If the called program possesses the INITIAL attribute, it and each program directly or indirectly contained within it, is placed into its initial state every time the called program is within a run unit.
10. Files associated with a called program's internal file connectors are not in the open mode when the program is in an initial state. Refer to "Initial State of a Program" on page 11-12.

On all other entries into the called program, the states and positioning of all such files are the same as when the called program was last exited.

11. The process of calling a program or exiting from a called program does not alter the status or positioning of a file associated with any external file connector.
12. The parameter values referenced in the CALL statement USING phrase are made available to the called program at the time the CALL statement is executed.
13. The CALL statement may appear anywhere within a segmented program. Therefore, when a CALL statement appears in a section with a segment number greater than or equal to 50, that segment remains in its current state when the EXIT PROGRAM statement returns control to the calling program.

14. *The BY CONTENT, BY REFERENCE and BY VALUE phrases are transitive across the parameters that follow them until they encounter another BY CONTENT, BY REFERENCE or BY VALUE phrase.* *MF*

If no BY CONTENT, BY REFERENCE or BY VALUE phrase is specified prior to the last parameter, the BY REFERENCE phrase is assumed.

-
15. If the BY REFERENCE phrase is specified or implied for a parameter, the object program operates as if the data item in the called program occupies the same memory area as the data item in the calling program. The description of the data item in the called program must specify the same number of character positions as specified by the description of the corresponding data item in the calling program.
- If a USING phrase specifies a BY REFERENCE ADDRESS OF record name, the address of the record is passed to the CALLED program. The corresponding item in the USING phrase of the CALLED program's Procedure Division header must have USAGE POINTER.* VSC2
16. If the BY CONTENT phrase is specified or implied for a parameter, the called program cannot change the value of this parameter. However, the called program may change the value of the data item referenced by the corresponding data-name in the called program's Procedure Division header.
- The data description for each parameter of the CALL statement's BY CONTENT phrase must be the same as the data description of the corresponding parameter in the USING phrase of the Procedure Division header. This means no conversion, extension, or truncation.
- If literal-2 is specified, its length must be the same as the corresponding parameter in the USING phrase of the Procedure Division header.* VSC2
- If LENGTH of identifier-3 is specified, the corresponding parameter in the Procedure Division header USING phrase should be defined as PIC 9(9) USAGE IS COMPUTATIONAL.*
17. *If the BY VALUE phrase is specified or implied for a parameter, the called program cannot change the value of this parameter in the calling program. Only the value of the parameter is passed to the called program. The address of the parameter is not passed. However, if the parameter is longer than 4 bytes, the address, instead of the value, is passed. The BY VALUE phrase is intended for use in calling programs in other languages. It should not be used in calling a COBOL program. Refer to the User's Guide for details of the format the value takes when it is passed to your system.* MF
18. Called programs may contain CALL statements. However, a called program must not execute a CALL statement that directly or indirectly calls the calling program. If a CALL statement is executed within the range of a declarative, that CALL statement cannot reference any called program to which control has been transferred and which has not completed execution.
19. The END-CALL phrase delimits the scope of the CALL statement.

Example 1

The following programs show the call by reference mechanisms. They are identical in meaning.

Calling Program Description

```
WORKING-STORAGE SECTION.  
01 PARAM-LIST.  
    05 PARTCODE PIC A.  
    05 PARTNO   PIC X(4).  
    05 U-SALES  PIC 9(5).  
    :  
    :  
PROCEDURE DIVISION.  
    :  
    :  
    CALL "CALLED-PROG"  
        USING PARAM-LIST.
```

Called Program Description

```
LINKAGE SECTION.  
01 USING-LIST.  
    10 PART-ID PIC X(5).  
    10 SALES   PIC 9(5).  
    :  
    :  
PROCEDURE DIVISION USING USING-LIST.
```

Calling Program Description

```
WORKING-STORAGE SECTION.  
01 PARAM-LIST.  
    05 PARTCODE PIC A.  
    05 PARTNO   PIC X(4).  
    05 U-SALES  PIC 9(5).  
    :  
    :  
PROCEDURE DIVISION.  
    :  
    :  
    CALL "CALLED-PROG"  
        USING BY REFERENCE PARAM-LIST.
```

Called Program Description

```
LINKAGE SECTION.  
01 USING-LIST.  
    10 PART-ID PIC X(5).  
    10 SALES   PIC 9(5).  
    :  
    :  
PROCEDURE DIVISION USING USING-LIST.
```

Example 2

In the following example, the called program cannot change the value of PARAM-LIST because it is called by value:

Calling Program Description

```
WORKING-STORAGE SECTION.  
01 PARAM-LIST.  
   05 PARTCODE PIC A.  
   05 PARTNO   PIC X(4).  
   05 U-SALES  PIC 9(5).  
   :  
   :  
PROCEDURE DIVISION.  
   :  
   :  
   CALL "CALLED-PROG" USING  
       BY CONTENT PARAM-LIST.
```

Called Program Description

```
LINKAGE SECTION.  
01 USING-LIST.  
   10 PART-ID PIC X(5).  
   10 SALES   PIC 9(5).  
   :  
   :  
PROCEDURE DIVISION USING USING-LIST.
```

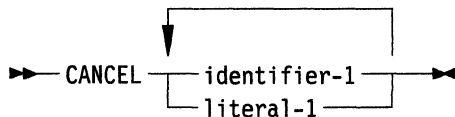
CANCEL Statement

Function

The CANCEL statement closes the named program and resets the program to its initial state before it is called next time.

General Format

The following figure shows the format of the CANCEL statement:



Syntax Rules

The following syntax rules apply to the CANCEL statement:

1. literal-1 must be a nonnumeric literal.
2. identifier-1 must reference an alphanumeric data item.

General Rules

The following rules apply to the CANCEL statement:

1. literal-1 or the content of the data item referenced by identifier-1 identifies the program to be cancelled.
2. The execution of a CANCEL statement ends any logical relationship between the program and the run unit. If a CANCELED program is called again, it is in its initial state. Refer to "Initial State of a Program" on page 11-12.
3. A program named in a CANCEL statement in another program must be callable by that other program. Refer to "Scope of Names" on page 11-8, and "CALL Statement" on page 11-36.
4. When an explicit or implicit CANCEL statement is executed, all programs contained within the program referenced by the CANCEL statement are also cancelled. The result is the same as if a valid CANCEL statement were executed for each contained program in the reverse order in which the programs appear in the containing program.
5. A program named in the CANCEL statement must not refer directly or indirectly to any called program that has not yet executed an EXIT PROGRAM statement.
6. A logical relationship to a cancelled program is established only by calling it again.
7. A called program is cancelled by being the operand of a CANCEL statement, by run unit termination, or by execution of an EXIT PROGRAM statement in a called program that possesses the INITIAL attribute.

-
8. A CANCEL statement naming a program not called in this run unit or called but cancelled is ignored. Control is transferred to the next executable statement following the explicit CANCEL statement.
 9. The contents of data items in external data records described by a program are not changed when that program is cancelled.
 10. During execution of a CANCEL statement, an implicit CLOSE statement without any optional phrases is executed for each file in the open mode associated with an internal file connector. Any USE procedures associated with any of these files are not executed.

CHAIN Statement

Function

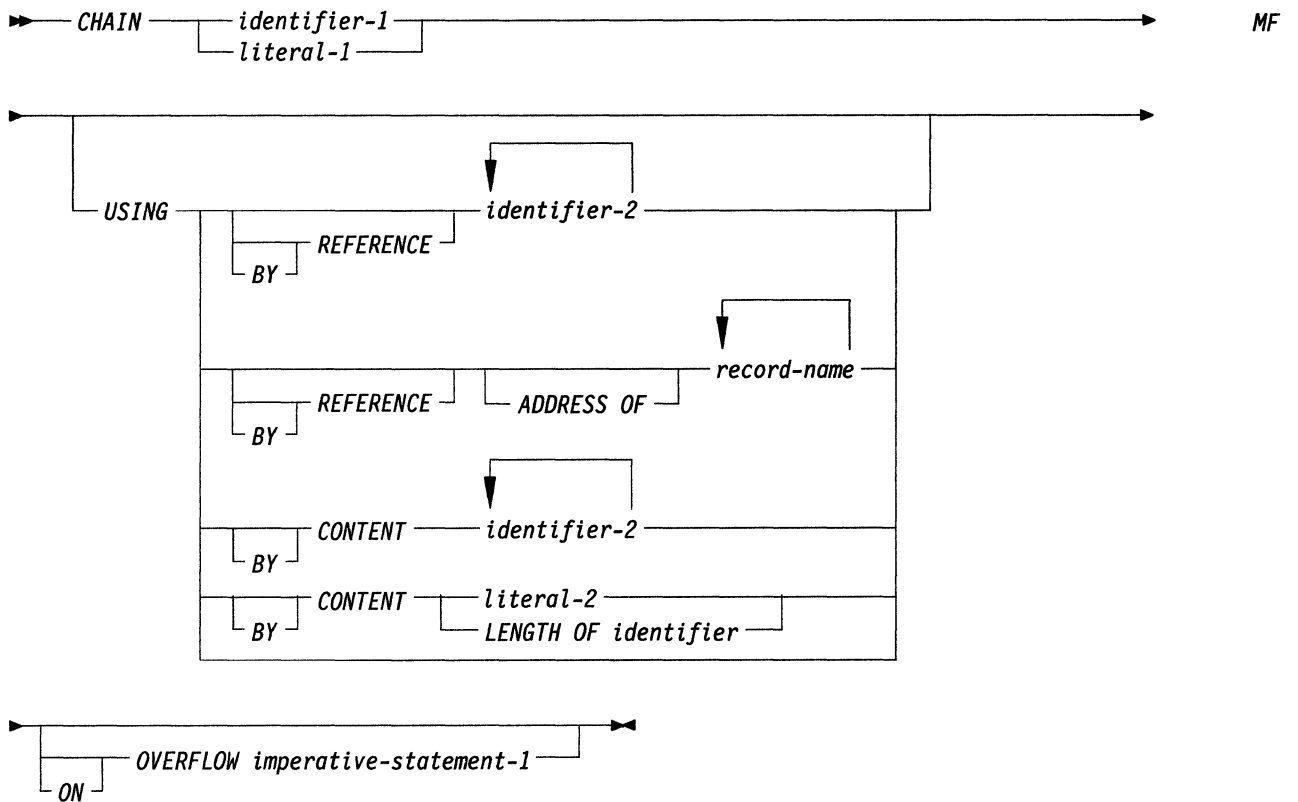
The CHAIN statement transfers control from one object program to another in the run unit. Control is not returned to the first object program.

MF

General Format

The following figure shows the format of the CHAIN statement:

MF



Syntax Rules

The following syntax rules apply to the CHAIN statement:

MF

1. *identifier-1 must be defined as an alphanumeric data item.*
2. *literal-1 must be a nonnumeric literal.*
3. *The USING phrase is included in the CHAIN statement only if there is a USING phrase in the Procedure Division header of the chained program. The number of operands in the two USING phrases can be unequal. Refer to "Procedure Division Header" on page 11-33.*
4. *Each of the operands in the USING phrase must have been defined as a data item in the FILE SECTION, WORKING-STORAGE SECTION, COMMUNICATION SECTION or LINKAGE SECTION. Each operand must have a level number of 01 through 49 or 77.*
5. *identifier-1, identifier-2, ... may be qualified when they reference data items defined in the FILE SECTION or the COMMUNICATION SECTION.*
6. *The Procedure Division parameters in the chained program must be declared in the WORKING-STORAGE SECTION, not in the LINKAGE SECTION. The parameters in the chaining program can be declared in any section.*
7. *literal-2 must be nonnumeric and cannot be a figurative constant.*
8. *Record name must be the name of the 01 or 77 level item in the linkage or WORKING-STORAGE SECTION.*

General Rules

The following rules apply to the CHAIN statement:

MF

1. *The execution of a CHAIN statement passes control to the chained program as the main program in a new run unit. After control is transferred, the chaining program and all other programs in its run unit are cancelled.*
2. *The chain operation moves the value from each data item referenced in the USING phrase of the CHAIN statement to the data item referenced in the USING phrase of the Procedure Division header of the chained program.*
3. *A chained program cannot be a nested program.*
4. *Chained programs may contain CHAIN statements. A chained program may chain directly or indirectly to the program from which it was chained.*
5. *If the chained program is not found, the run is abandoned with a Run Time Environment error message.*
6. *If the chained program is too big to fit in memory, the run is abandoned with a Run Time Environment error message.*

ENTRY Statement

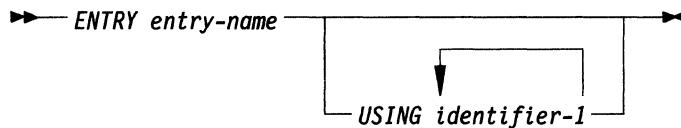
Function

The *ENTRY* statement establishes an alternate entry-point into a called OSVS VSC2 COBOL program.

General Format

The following figure shows the format of the *ENTRY* statement:

OSVS VSC2



Syntax Rules

The following syntax rules apply to the *ENTRY* statement:

OSVS VSC2

1. Entry name may be any alphanumeric literal.
2. The entry name must not be the name of the called program in which it appears, but it must follow the same rules of formation.
3. An entry point cannot be used in a nested program.

DBCS Support

4. *identifier-1* may be a DBCS literal.

End of DBCS Support

General Rules

The following rules apply to the ENTRY statement:

OSVS VSC2

1. When a CALL statement names an alternate entry-point in a calling program, control is transferred to the next executable statement following the ENTRY statement in the called program.
2. The called program can be a program in an external file that has previously been called using its program-id. The AIX VS COBOL system must be able to locate the program containing the entry-point when the entry-point is named. Refer to the User's Guide.
3. Refer to "Procedure Division Header" on page 11-33 for information on the USING phrase and rules for definition of identifier-1, etc..

Example

The following example shows a called program with an ENTRY statement:

OSVS VSC2

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. CALLSTAT.  
:  
:  
DATA DIVISION.  
:  
:  
WORKING-STORAGE SECTION.  
01 RECORD-2 PIC X.  
01 RECORD-1.  
   05 SALARY      PICTURE S9(5)V99.  
   05 RATE        PICTURE S9V99.  
   05 HOURS       PICTURE S99V9.  
:  
:  
PROCEDURE DIVISION.  
:  
:  
   CALL "SUBPROG" USING RECORD-1.  
:  
:  
   CALL "PAYMASTER" USING RECORD-1 RECORD-2.  
:  
:  
   STOP RUN.
```

IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROG.
:
:
DATA DIVISION.
:
:
LINKAGE SECTION.
01 PAYREC.
10 PAY PICTURE S9(5)V99.
10 HOURLY-RATE PICTURE S9V99.
10 HOURS PICTURE S99V9.
77 CODE PIC 9.
:
:
PROCEDURE DIVISION USING PAYREC.
:
:
EXIT PROGRAM.
ENTRY "PAYMASTER" USING PAYREC CODE.
:
:
GOBACK.

EXIT PROGRAM Statement

Function

The EXIT PROGRAM statement marks the logical end of a called program.

General Format

The following figure shows the format of the EXIT PROGRAM statement:

▶— EXIT PROGRAM. —▶

Syntax Rules

The following syntax rules apply to the EXIT PROGRAM statement:

1. The EXIT PROGRAM statement should appear in a sentence by itself.
This rule is not enforced. *MF*
2. The EXIT PROGRAM sentence must be the only sentence in the paragraph.
This rule is not enforced. *MF*
3. The EXIT PROGRAM statement must not appear in a declarative procedure in which the GLOBAL phrase is specified.

General Rules

The following rules apply to the EXIT PROGRAM statement:

1. If the EXIT PROGRAM statement is executed in a program not under the control of a calling program, the statement continues execution of the program with the next executable statement.
2. The execution of an EXIT PROGRAM statement in a called program without the initial attribute, continues execution with the next executable statement following the CALL statement in the calling program. The state of the calling program is not altered from when it executed the CALL statement except that the contents of data items and the contents of data files shared between the calling and called program may have changed. The state of the called program is not altered except that the end range of all PERFORM statements have been reached.
3. Other than the actions specified in general rule 2, the execution of an EXIT PROGRAM statement in a called program that possesses the initial attribute is the same as executing a CANCEL statement. Refer to "CANCEL Statement" on page 11-43.

GOBACK Statement

Function

The GOBACK statement marks the logical end of a called program.

OSVS VSC2

General Format

The following figure shows the format of the GOBACK statement:

OSVS VSC2

▶— GOBACK. —▶

Syntax Rule

A GOBACK statement must be the only statement, or the last of a series of imperative statements, in a sentence.

OSVS VSC2

General Rules

The following rules apply to the GOBACK statement:

OSVS VSC2

- 1. If program execution reaches a GOBACK statement while operating under control of a calling program, control returns to the point immediately after the CALL statement in the calling program.*
- 2. If program execution reaches a GOBACK statement in a main program, the GOBACK statement terminates the program, as in the STOP RUN statement. Refer to Chapter 3, "Introduction to the Nucleus"*
- 3. The user should be aware that if neither the OSVS nor the VSC2 option is given, then the GOBACK verb will appear syntactically to be a user-defined paragraph name. This might cause some unexpected changes in the flow of control that are difficult to detect.*

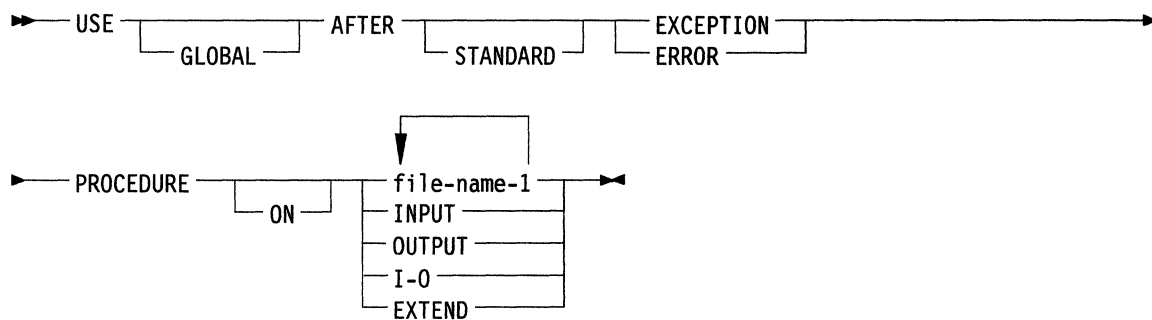
USE Statement

Function

Within the Interprogram Communication module, the USE statement determines the associated declarative procedures invoked during the execution of any program contained within the program with the USE statement.

General Format

The following figure shows the format of the USE statement:



Syntax Rule

Refer to Chapter 8, "File Input and Output."

General Rules

Special precedence rules are followed when programs are contained within other programs. In applying these rules, only the first qualifying declarative is selected for execution. The declarative selected for execution must satisfy the rules for execution of that declarative. The order of precedence for selecting a declarative is:

1. The declarative in the program containing the statement that caused the qualifying condition.
2. The declarative in which the GLOBAL phrase is specified and which is within the program directly containing the program which was last examined for a qualifying declarative.
3. Any declarative selected by applying rule 2 to each more inclusive containing program until rule 1 is applied to the outermost program. If no qualifying declarative is found, none is executed.

USE BEFORE REPORTING Statement

Function

Within the Interprogram Communication module, the USE BEFORE REPORTING statement determines whether the associated declarative procedures are invoked during the execution of any program contained within the program that includes the USE BEFORE REPORTING statement.

General Format

The following figure shows the format of the USE BEFORE REPORTING statement:



General Rules

Special precedence rules are followed when programs are contained within other programs. In applying these rules, only the first qualifying declarative is selected for execution. The declarative selected for execution must satisfy the rules for execution of that declarative. The order of precedence for selecting a declarative is:

1. The declarative within the program that contains the statement which caused the qualifying condition.
2. The declarative in which the GLOBAL phrase is specified and which is within the program directly containing the program last examined for a qualifying declarative.
3. Any declarative selected by applying rule 2 to each more inclusive containing program until rule 1 is applied to the outermost program. If no qualifying declarative is found, none is executed.

PART 4. Advanced Features

Chapter 12. Table-Handling

Contents

About This Chapter	12-3
Introduction	12-4
Data Division in the Table-Handling Module	12-5
OCCURS Clause	12-5
Function	12-5
General Format	12-5
Syntax Rules	12-6
General Rules	12-7
Example 1	12-8
Example 2	12-9
Example 3	12-10
Example 4	12-10
USAGE IS INDEX Clause	12-12
Function	12-12
General Format	12-12
Syntax Rules	12-12
General Rules	12-12
Procedure Division in the Table-Handling Module	12-13
Relation Condition—Comparisons Involving Index-Names and/or Index Data-Items	12-13
Overlapping Operands	12-13
SEARCH Statement	12-14
Function	12-14
General Format	12-14
Syntax Rules	12-15
General Rules	12-16
SET Statement	12-19
Function	12-19
General Format	12-19
Syntax Rules	12-19
General Rules	12-19
Table-Handling Sample Program	12-21

About This Chapter

This chapter describes the Table-Handling module. The following aspects are covered:

- Defining a table
- Accessing a table
- Subscripting and indexing
- Determining maximum dimensions of a table.

Introduction

The table-handling module defines tables of contiguous data-items and accesses an item relative to its position in the table. Language facilities specify how many times an item is to be repeated. Each item may be identified through use of a subscript or an index. Refer to Chapter 2, “COBOL Concepts.”

Table-handling allows accessing of items in variable-length tables and multiple dimensions. The maximum number of multiple dimensions is seven.

Sixteen dimensions are permitted.

MF

In addition, table-handling allows specifying of ascending or descending keys and permits searching a dimension of a table for an item satisfying a specified condition.

In COBOL, a table is defined with an OCCURS clause in its data description. The OCCURS clause specifies that the named item is to be repeated as many times as stated. The named item is considered a **table element**, and its name and description apply to each repetition (or occurrence) of the item. Since the occurrences are not given unique data-names, reference to a particular occurrence can be made only by specifying the data-name of the table element, together with the occurrence number of the desired item within the element.

The occurrence number is known as a **subscript**. The technique of supplying the occurrence number of individual table elements is called **subscripting**. A related technique, called **indexing**, is also available for table references. Both subscripting and indexing are described in the following sections.

Data Division in the Table-Handling Module

This section describes the Data Division in the table-handling module.

OCCURS Clause

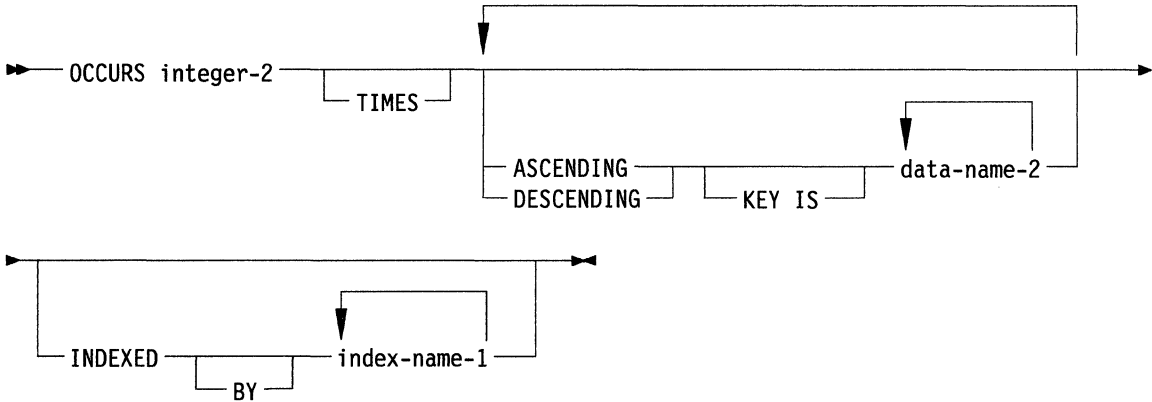
Function

The OCCURS clause eliminates the need for separate entries for repeated data-items and supplies information required for the application of subscripts or indices.

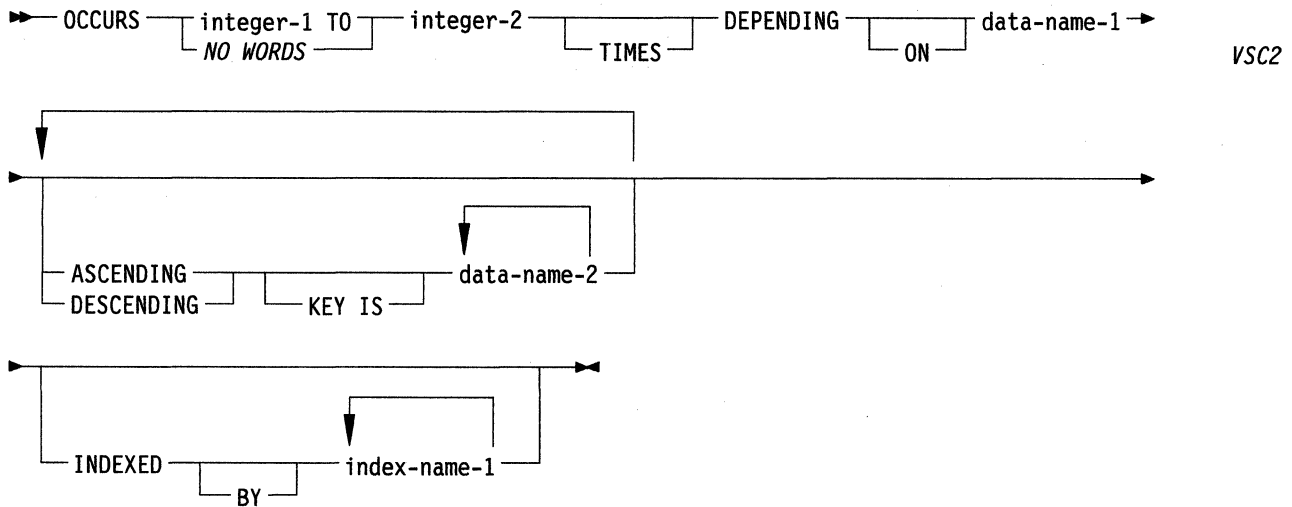
General Format

The following figures show the general format for the OCCURS clause:

Format 1



Format 2



Syntax Rules

The following rules apply to the OCCURS clause:

1. Where both integer-1 and integer-2 are used, the value of integer-1 must be less than the value of integer-2. *In Format 2, integer-1 TO may be omitted. In this case the default value one is assumed.* VSC2
2. The data description of data-name-1 must describe a positive integer or zero.
3. data-name-1, data-name-2, data-name-3, ... may be qualified.
4. data-name-2 must be the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause.
5. Additional instances of data-name-2 must name an entry subordinate to the group item which is the subject of this entry.
6. An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referred to by indexing. The index-name identified by this clause is not defined elsewhere, and not being data, cannot be associated with any data hierarchy.
7. A data description entry containing Format 2 of the OCCURS clause should only be followed, within the record description, by data description entries subordinate to it.

A data description entry containing Format 2 of the OCCURS clause may be followed, within the record description, by data description entries which are not subordinate to it. The positions of these entries within the record varies at run time with the value of the data-item referenced in the DEPENDING ON clause unless the NOODOSLIDE system directive is set. In this case, the containing record is considered as always containing the maximum number of occurrences of the Format 2 item, irrespective of the value of data-name-1. If the ODOSLIDE directive is set, when the value of data-name-1 is changed, the position referenced by identifiers following, but not subordinate to the table are dynamically changed. The data these items contain may be lost.

OSVS

MF

-
8. The OCCURS clause cannot be specified in a data description entry that:
 - a. Has 66 or 88 level-number
 - b. Describes an item whose size is variable. The size of an item is variable if the data description of any subordinate item contains Format 2 of the OCCURS clause.
 9. The OCCURS clause should not be specified in a data description entry at the 01 level or as a 77 level-number. *This restriction may be ignored.* *MF*
 10. In Format 2, the data-item defined by data-name-1 must not occupy a character position between the first character position defined by the data description entry and the last character position defined by the record description entry.
 11. If data-name-2 is not the subject of this entry, then:
 - a. All of the items identified by the data names in the KEY IS phrase must be within the group item which is the subject of this entry.
 - b. Items identified by the data name in the KEY IS phrase must not contain an OCCURS clause.
 - c. There must not be any entry that contains an OCCURS clause between the items identified by the data names in the KEY IS phrase and the subject of this entry.
 12. index-name-1 must be a unique word within the program.
 13. If the OCCURS clause is specified in a data description entry for a record description entry containing the EXTERNAL clause, then data-name-1 (the DEPENDING ON variable) must refer to a data item also having the EXTERNAL attribute.
 14. If the OCCURS clause is specified in a data description entry for a record description entry containing the GLOBAL clause, then data-name-1 (the DEPENDING ON variable) must refer to a data item also having the GLOBAL attribute.

DBCS Support

15. The OCCURS clause may be specified for a Double-Byte Character Set (DBCS) item.
16. The ASCENDING/DESCENDING KEY phrase (for a SEARCH ALL statement only) can be specified in the OCCURS clause for a DBCS item.

End of DBCS Support

General Rules

The following rules apply to the OCCURS clause:

1. The OCCURS clause is used in defining tables and other homogenous sets of repeated data-items. Whenever the OCCURS clause is used, the data name which is the subject of this entry must be subscripted or indexed when referred to in a statement other than SEARCH or USE FOR DEBUGGING. Further, if the subject of this entry is the name of a group item, then all data names belonging to the group must be subscripted or indexed when they are used as operands, except as the object of a REDEFINES clause. Refer to "Subscripting" on page 2-30, "Indexing" on page 2-31, and "Identifier" on page 2-33.
2. When a description includes an OCCURS clause, all description clauses, except for the OCCURS clause, apply to each occurrence of the item described. Refer to the restriction in rule 1b on page 6-44, under "Data Description Entries Other Than CONDITION-NAMES and CONSTANT-NAMES" on page 6-44.
3. The number of occurrences of the subject entry is defined as follows:
 - a. In Format 1, the value of integer-2 represents the exact number of occurrences.

-
- b. In Format 2, the current value of the data-item referenced by data-name-1 represents the number of occurrences.
 This format specifies a variable number of occurrences for the subject of this entry. The value of integer-2 represents the maximum number of occurrences. The value of integer-1 represents the minimum number of occurrences. This does not mean that the length of the subject is variable, but that the number of occurrences is variable.
 The value of the data-item referenced by data-name-1 must fall in the range of integer-1 through integer-2. Reducing the value of the data-item referenced by data-name-1 makes the contents of data-items unpredictable. The data-items occurrence numbers exceed the value of the data-item referenced by data-name-1.
 4. When a group item, with a subordinate entry specifying Format 2 of the OCCURS clause, is referenced, the part of the table area used in the operation is determined as follows:
 - a. If the data-item referenced by data-name-1 is outside the group, only the table area specified by the value referenced by data-name-1 will be used.
 - b. If the data-item referenced by data-name-1 is in the same group and the group data-item is referenced as a sending item, only the table area specified by the value referenced by data-name-1 will be used in the operation. If the group is a receiving item, the maximum length of the group will be used.
 5. The KEY IS phrase indicates that the repeated data is arranged in ascending or descending order according to the value contained in data-name-2. The ascending or descending order is determined by the rules for comparison of operands. Refer to "Comparison of Numeric Operands" on page 7-12, and "Comparison of Nonnumeric Operands" on page 7-12. The data names are listed in their descending order of significance.

Example 1

The following examples show single- and multiple-dimensional tables:

```
01 TABLE-ONE.
   05 ELEMENT-ONE OCCURS 3 TIMES.
     10 ELEMENT-A PIC X(4).
     10 ELEMENT-B PIC 9(4).
```

TABLE-ONE is the group item that contains the table. ELEMENT-ONE names the table element of a one-dimensional table that occurs 3 times. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-ONE.

```
01 TABLE-TWO.
   05 ELEMENT-ONE OCCURS 3 TIMES.
     10 ELEMENT-TWO OCCURS 3 TIMES.
       15 ELEMENT-A PIC X(4).
       15 ELEMENT-B PIC 9(4).
```

ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE.

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

In this example, ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-THREE is an element of a

three-dimensional table that occurs two times within each occurrence of ELEMENT-TWO. See Table 12-1 on page 12-9 for an explanation of storage layout for Table-Three.

Table 12-1. Storage Layout for Table-Three		
Element-One Occurs 3 Times	Element-Two Occurs 3 Times	Element-Three Occurs 2 Times
Element-one (1)	Element-two (1 1) Element-two (1 2) Element-two (1 3)	Element-three (1 1 1) Element-three (1 1 2) Element-three (1 2 1) Element-three (1 2 2) Element-three (1 3 1) Element-three (1 3 2)
Element-one (2)	Element-two (2 1) Element-two (2 2) Element-two (2 3)	Element-three (2 1 1) Element-three (2 1 2) Element-three (2 2 1) Element-three (2 2 2) Element-three (2 3 1) Element-three (2 3 2)
Element-one (3)	Element-two (3 1) Element-two (3 2) Element-two (3 3)	Element-three (3 1 1) Element-three (3 1 2) Element-three (3 2 1) Element-three (3 2 2) Element-three (3 3 1) Element-three (3 3 2)

Example 2

This example shows a program using a multi-dimensional table with INDEXED BY option. Each *index-name* identifies an index to be used in table references. The index-name is specified through the OCCURS clause.

Each index named is a compiler-generated storage area. It contains a binary value representing an actual displacement from the beginning of the table element. To be valid, the displacement value must correspond to the value of an occurrence number in the table element. Two forms of indexing are provided: direct and relative.

In *direct indexing*, the index-name is in the form of a subscript. The value contained in the index is then calculated as the occurrence number minus one, multiplied by the length of the individual table entry. For example:

```
05 ELEMENT-A OCCURS 10 INDEXED BY INX-A PIC X(8).
```

For the fifth occurrence of ELEMENT-A, the binary value contained in INX-A is $(5 - 1) * 8 = 32$.

In *relative indexing*, the index-name is followed by a space, followed by a + or a -, followed by another space, followed by an unsigned numeric literal. The literal is considered to be an occurrence number, and is converted to an index value before being added to or subtracted from the index-name index.

The following example shows indexing specified for Table 12-1.

```
01 TABLE-THREE.
05 ELEMENT-ONE OCCURS 3 TIMES INDEXED BY INX-1.
10 ELEMENT-TWO OCCURS 3 TIMES INDEXED BY INX-2.
15 ELEMENT-THREE OCCURS 2 TIMES INDEXED BY INX-3
   PICTURE X(8).
```

Then a relative index reference to

ELEMENT-THREE (INX-1 + 1, INX-2 + 2, INX-3 -1)

computes the displacement:

(address of ELEMENT-ONE)
+ (contents of INX-1) + (48 * 1)
+ (contents of INX-2) + (16 * 2)
+ (contents of INX-3) - (8 * 1)

(Each occurrence of ELEMENT-ONE is 48 characters in length. Each occurrence of ELEMENT-TWO is 16 characters in length. Each occurrence of ELEMENT-THREE is 8 characters in length.)

Example 3

The following is an example of a table INDEXED BY ASCENDING/DESCENDING KEY option:

```
WORKING-STORAGE SECTION.  
01 TABLE-RECORD.  
    05 EMPLOYEE-TABLE OCCURS 100 TIMES  
        ASCENDING KEY IS WAGE-RATE EMPLOYEE-NO  
        INDEXED BY A, B.  
        10 EMPLOYEE-NAME          PIC X(20).  
        10 EMPLOYEE-NO            PIC 9(6).  
        10 WAGE-RATE              PIC 9999V99.  
    10 WEEK-RECORD OCCURS 52 TIMES  
        ASCENDING KEY IS WEEK-NO INDEXED BY C.  
        15 WEEK-NO                PIC 99.  
        15 AUTHORIZED-ABSENCES    PIC 9.  
        15 UNAUTHORIZED-ABSENCES PIC 9.  
        15 LATENESS               PIC 9.
```

The keys for EMPLOYEE-TABLE are subordinate to that entry. The key for WEEK-RECORD is subordinate to that subordinate entry.

Note that keys must be listed in decreasing order of significance.

Example 4

The following are examples of the initialization of a table:

```
01 TABLE-ONE  
    05 ELEMENT-ONE PICTURE X VALUE "1".  
    05 ELEMENT-TWO PICTURE X VALUE "2".  
    05 ELEMENT-THREE PICTURE X VALUE "3".  
    05 ELEMENT-FOUR PICTURE X VALUE "4".  
01 TABLE-TWO REDEFINES TABLE-ONE.  
    05 OCCURS-ELEMENT OCCURS 4 TIMES PICTURE X.
```

The table can be described as a record containing contiguous subordinate data description entries. Each entry contains a VALUE clause for the initial value. The record is then redescribed through a REDEFINES entry containing a subordinate entry with an OCCURS clause. Due to the OCCURS clause, the subordinate entries of the redefined entry are repeated.

```
01 TABLE-ONE VALUE "1234".  
    05 TABLE-TWO OCCURS 4 TIMES PICTURE X.
```

If the subordinate entries do not require separate handling, the VALUE of the entire entry can be specified in the entry which names the table. The lower level entries then contain OCCURS clauses, and show the hierarchical structure of the table. The subordinate entries must not contain VALUE clauses.

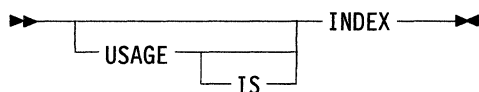
USAGE IS INDEX Clause

Function

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

General Format

The following figure shows the general format for the USAGE IS INDEX clause:



Syntax Rules

The following syntax rules apply to the USAGE IS INDEX clause:

1. An index data-item can be referenced explicitly only in one of the following:
 - A SEARCH or SET statement
 - A relation condition
 - The USING phrase of a Procedure Division header
 - The USING phrase of a CALL statement.
2. The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

General Rules

The following general rules apply to the USAGE IS INDEX clause:

1. The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.
2. An elementary item described with the USAGE IS INDEX clause is called an index data-item and contains a value which must correspond to an occurrence number of a table element. The elementary item cannot be a conditional variable. The VS COBOL system allocates a 4-byte binary field with an implied PICTURE of 9(9) COMP. If the USAGE IS INDEX clause describes a group item, the elementary items in the group are all index data-items. The group itself is not an index data-item and cannot be used in the SEARCH or SET statement or in a relation condition.
3. An index data-item may be part of a group referred to in a MOVE or input-output statement, in which case no conversion takes place.

Procedure Division in the Table-Handling Module

This section describes the Procedure Division in the table-handling module.

Relation Condition—Comparisons Involving Index-Names and/or Index Data-Items

Relation tests may be made between the following data-items:

- Two index-names. The result is the same as if the corresponding occurrence numbers were compared.
- An index-name and a data-item (other than an index data-item) or literal. The occurrence number corresponding to the value of the index-name is compared to the data-item or literal.
- An index data-item and an index-name or another index data item. The actual values are compared without conversion.
- The result of the comparison of an index data-item with any data-item or literal not specified above is undefined.

Overlapping Operands

When a sending and a receiving item in a SET statement share a part of their storage area, the result of the execution of such a statement is undefined.

SEARCH Statement

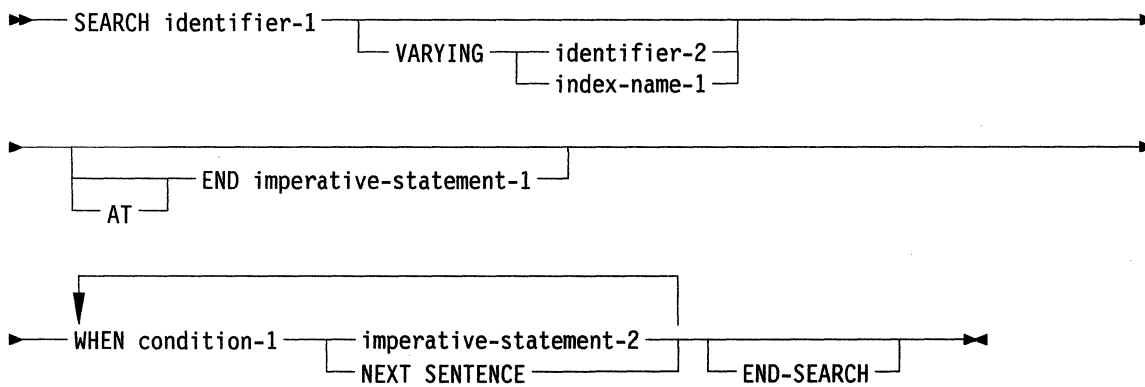
Function

The SEARCH statement searches for a table element satisfying the specified condition and adjusts the associated index name to indicate that table element.

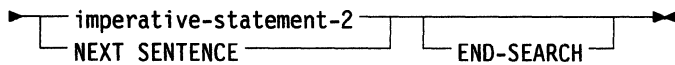
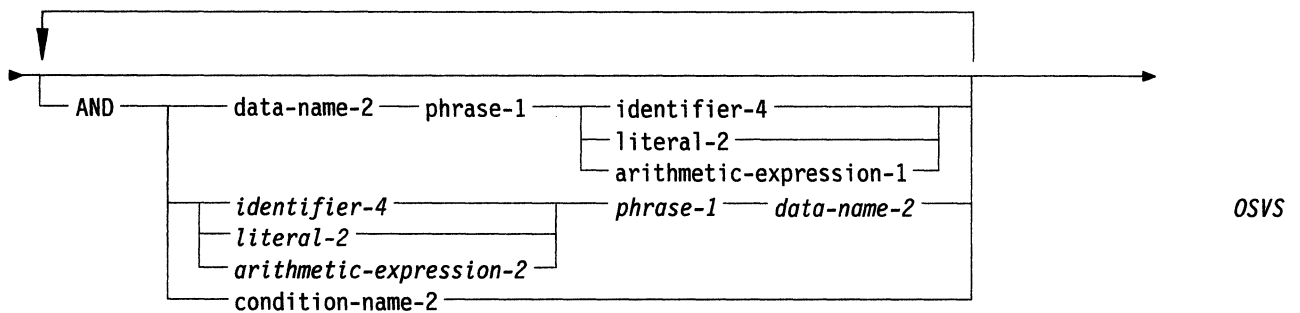
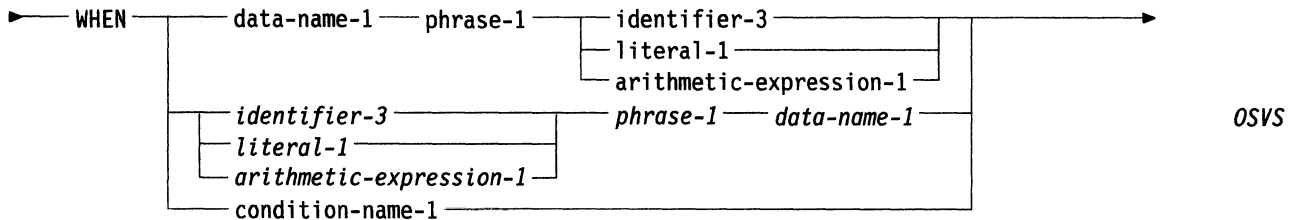
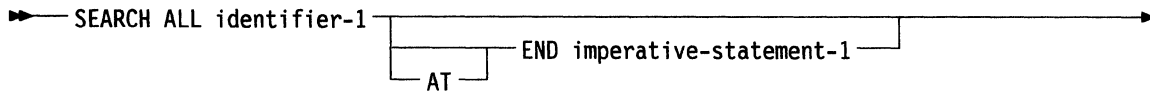
General Format

The following figures show the general format for the SEARCH statement:

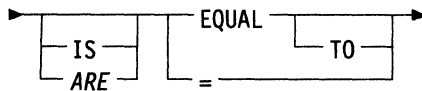
Format 1



Format 2



where phrase-1 is:



OSVS

Syntax Rules

The following rules apply to the SEARCH statement:

1. In both Formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause. The description of identifier-1 in Format 2 must also contain the KEY IS phrase in its OCCURS clause.
2. identifier-2, when specified, must be described as USAGE IS INDEX or as a numeric elementary item without any positions to the right of the assumed decimal point.
3. In Format 1, condition-1 may be any condition as described in "Conditional Expressions" on page 7-9.
4. In Format 2, all referenced condition-names must be defined as having only a single value. The data-name associated with a condition-name must appear in the KEY clause of identifier-1. Each data-name-1, data-name-2 may be qualified. Each data-name-1, data-name-2 must be indexed by the first index-name associated with identifier-1 along with other indices or literals as required. Each data-name-1, data-name-2 must also be referenced in the KEY clause of identifier-1. identifier-3, identifier-4, or identifiers specified in arithmetic-expression-1, arithmetic-expression-2

must not be referenced in the KEY clause of identifier-1 or be indexed by the first index-name associated with identifier-1.

In Format 2, when a data-name in the identifier-1 KEY clause is referenced, or when a condition-name associated with a data-name in the identifier-1 KEY clause is referenced, all preceding data-names or associated condition-names must also be referenced.

5. If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase must not be specified.

DBCS Support

6. identifier-1 can be a DBCS (USAGE DISPLAY-1) item. (Format 1)
7. condition-1 may include DBCS relations and/or DBCS condition-name conditions.
8. identifier-1 can be a DBCS item if the ASCENDING/DESCENDING KEY is defined as a DBCS item. (Format 2)

End of DBCS Support

General Rules

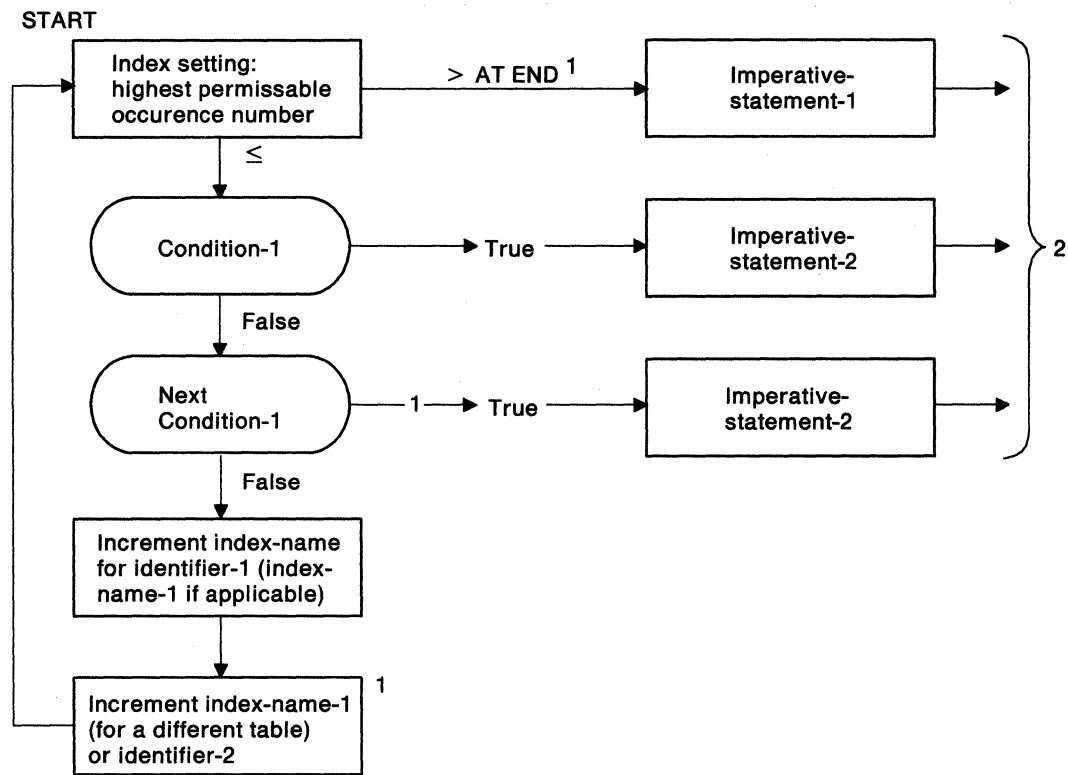
The following general rules apply to the SEARCH statement:

1. The scope of a SEARCH statement may be terminated by any of the following:
 - a. An END-SEARCH phrase at the same level of nesting
 - b. A separator period
 - c. An ELSE or END-IF associated with a previous IF statement.
2. If Format 1 of the SEARCH is used, a serial type of search operation takes place, starting with the current index setting.
 - a. The SEARCH terminates immediately if, at the start, the index-name for identifier-1 contains a value corresponding to an occurrence number greater than the highest permissible occurrence number for identifier-1. The number of occurrences of identifier-1, the last of which is the highest permissible, is discussed in the OCCURS clause. Refer to "OCCURS Clause" on page 12-5. If the AT END phrase is specified, imperative-statement-1 is executed. If the AT END phrase is not specified, control passes to the next executable sentence.
 - b. If the index-name for identifier-1 contains a value corresponding to an occurrence number in the permissible range for identifier-1, the SEARCH statement evaluates the conditions in the order they are written, making use of the index settings to determine the occurrence of those items to be tested. (The permissible range for identifier-1 is discussed in "OCCURS Clause" on page 12-5.)

If none of the conditions are satisfied, the index name for identifier-1 is incremented to reference the next occurrence. The process is then repeated using the new index-name settings unless the new value of the index-name settings for identifier-1 corresponds to a table element outside the permissible range of occurrence values, in which case the search terminates as indicated in rule 1a. If one of the conditions is satisfied upon its evaluation, the search terminates immediately and the imperative statement associated with that condition is executed. The index-name remains set at the occurrence which satisfied the condition.

-
3. In a Format 2 SEARCH, the results of the SEARCH ALL operation are predictable only when:
 - a. The data in the table is ordered in the same manner described in the ASCENDING/DESCENDING KEY clause, associated with the description of identifier-1, and
 - b. The contents of the key(s) referenced in the WHEN clause are sufficient to identify a unique table element.
 4. If Format 2 of the SEARCH is used, a nonserial type of search operation may take place. The initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation. The setting is varied with the restriction that it is never set to a value less than or greater than the range of values corresponding to the first and last element of the table. The length of the table is discussed in the OCCURS clause. If any of the conditions specified in the WHEN clause cannot be satisfied for any setting of the index within the permitted range, control is passed to imperative-statement-1 of the AT END phrase, if specified, or to the next executable sentence when this phrase is not specified. In either case the final setting of the index is not predictable. If all conditions can be satisfied, the index indicates an occurrence allowing the conditions to be satisfied, and control passes to imperative-statement-2.
 5. After execution of imperative-statement-1 or imperative-statement-2 that does not terminate with a GO TO statement, control passes to the next executable sentence.
 6. In Format 2, the index-name used for the search operation is the first index-name appearing in the identifier-1 INDEXED BY phrase. Any other index-names for identifier-1 remain unchanged.
 7. In Format 1, if the VARYING phrase is not used, the index-name used for the search operation is the first index-name appearing in the identifier-1 INDEXED BY phrase. Any other index-names for identifier-1 remain unchanged.
 8. In Format 1, if the VARYING index-name-1 phrase is specified and if index-name-1 appears in the identifier-1 INDEXED BY phrase, that index-name is used for this search. If this is not the case, or if the VARYING identifier-2 phrase is specified, the first index-name given in the identifier-1 INDEXED BY phrase is used for the search. In addition, the following operations occur:
 - a. If the VARYING index-name-1 phrase is used and if index-name-1 appears in the INDEXED BY phrase of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the index-name for identifier-1.
 - b. If the VARYING identifier-2 phrase is specified and identifier-2 is an index data-item, then the data-item referenced by identifier-2 is incremented by the same amount as, and at the same time as, the index for identifier-1. If identifier-2 is not an index data-item, the data-item referenced by identifier-2 is incremented by the value (1) at the same time as the index referenced by the index-name for identifier-1.
 9. If identifier-1 is a data-item subordinate to a data item containing an OCCURS clause (specifying a two or three dimensional table) an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data-item identifier-2 or index-name-1) is modified by the execution of the SEARCH statement. To search an entire two or three dimensional table, it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed to adjust index-names to appropriate settings.

Figure 12-1 shows a flowchart of the Format 1 SEARCH operation containing two WHEN phrases.



- ¹ These operations are options included only when specified in the SEARCH statement.
- ² Each of these transfers control to the next executable sentence unless the imperative-statement ends with a GO TO statement.

Figure 12-1. Flowchart of SEARCH Operation Containing Two WHEN Phrases

SET Statement

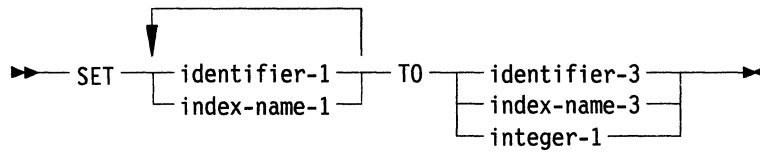
Function

The SET statement establishes reference points for table-handling operations by setting index-names for table elements.

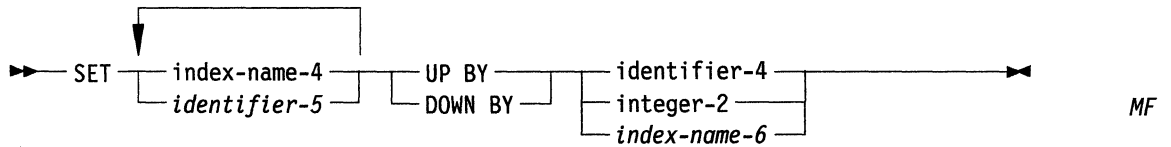
General Format

The following examples show the general format for the SET statement:

Format 1



Format 2



Syntax Rules

The following syntax rules apply to the SET statement:

1. identifier-1, identifier-3, and identifier-5 must name either index data-items or elementary items described as integers.
2. identifier-4 must be described as an elementary numeric integer.
3. integer-1 and integer-2 may be signed. integer-1 must be positive.

General Rules

The following general rules apply to the SET statement:

1. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

2. If index-name-3 is specified, the value of the index before the execution of the SET statement must correspond to an occurrence number of an element in the table. If index-name-4, index-name-5 is specified, the value of the index both before and after the execution of the SET statement must correspond to an occurrence number of an element in the table. If index-name-1 is specified, the value of the index after the execution of the SET statement must correspond to an occurrence number of an element in the table. The value of the index associated with an index-name after the execution of a SEARCH or PERFORM statement may be undefined. Refer to "SEARCH Statement" on page 12-14 and "PERFORM Statement" on page 7-73.
3. In Format 1, the following action occurs:
 - a. index-name-1 is set to a value that refers to the table element corresponding in occurrence number to the table element referenced by index-name-3, identifier-3, or integer-1. If identifier-3 is an index data-item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place.
 - b. If identifier-1 is an index data-item, it may be set equal to either the contents of index-name-3 or identifier-3 where identifier-3 is also an index item. No conversion takes place in either case.
 - c. If identifier-1 is not an index data-item, it may be set only to an occurrence number corresponding to the value of index-name-3. Neither identifier-3 nor integer-1 can be used in this case.
 - d. The process is repeated for any additional occurrences of index-name-1 or identifier-1. Each time the value of index-name-3 or identifier-3 is used as it was at the beginning of the statement's execution. Any subscripting or indexing for identifier-1, etc., is evaluated immediately before the value of the respective data-item is changed.
4. In Format 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value corresponding to the number of occurrences represented by the value of integer-2 or identifier-4. Thereafter, the process is repeated for any additional occurrences of index-name-4. Each time the value of identifier-4 is used as it was at the beginning of the execution of the statement.
5. Data in Table 12-2 represents the validity of various operand combinations in the SET statement. The numbers and letters in the receiving item list indicate the applicable general rule.

Table 12-2. SET Statement Valid Operand Combinations			
Sending Item	Receiving Item ¹		
	Integer Data-Item	Index-Name	Index Data-Item
Integer Literal	No/3c	Valid/3a	No/3b
Integer Data-Item	No/3c	Valid/3a	No/3b
Index-Name	Valid/3c	Valid/3a	Valid/3b ²
Index Data-Name	No/3c	Valid/3a	Valid/3b ²
Note:			
¹ = Rule numbers referred to in General Rules above ² = No conversion takes place.			

Table-Handling Sample Program

This program illustrates one method of building a table, as well as two methods of searching a table.

A chain store doing business in the continental United States divides the country into seven sales areas. A weekly report of sales is made for each area. The program places these sales figures into a cumulative table, and then issues a report comparing sales by area for the current week with those for the preceding week, as well as for the same week in the preceding year. Optional inquiries can be made to report—for any week of the current year—which areas increased sales over last year.

The following table-handling features are illustrated: the OCCURS clause, the OCCURS DEPENDING ON clause, the INDEXED BY option, direct indexing, relative indexing, the PERFORM UNTIL statement used to search a table element, the SEARCH VARYING STATEMENT used to search two table elements simultaneously, the SET TO statement, and the SET UP BY statement.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. TABLES.  
DATE-COMPILED. MAR 20, 1988.  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER. IBM-RISC-6000.  
OBJECT-COMPUTER. IBM-RISC-6000.
```

```
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT CURRENT-FILE  
        ASSIGN TO "CURRENT.FIL".  
    SELECT YR-TO-DATE-FILE  
        ASSIGN TO "DATE.FIL".  
    SELECT LAST-YR-FILE  
        ASSIGN TO "LASTYR.FIL".  
    SELECT PRINT-FILE  
        ASSIGN TO "PRINT.FIL".  
    SELECT OPTIONAL INQUIRY-FILE  
        ASSIGN TO "INQUIRY.FIL".
```

```
DATA DIVISION.  
FILE SECTION.  
FD CURRENT-FILE LABEL RECORDS OMITTED.  
01 CURRENT-SALE-RECORD.  
    05 CURR-WEEK PIC 99.  
    05 CURR-FIGURES OCCURS 7 TIMES INDEXED BY WEEKC.  
        10 CURR-AREA-NO PIC 9.  
        10 CURR-AREA-SALE PIC S99999V99.
```

```
FD YR-TO-DATE-FILE LABEL RECORDS STANDARD.  
01 YR-TO-DATE-RECORD.  
    05 YR-TO-DATE-TABLE OCCURS 52 TIMES INDEXED BY WEEKL.  
        10 FILLER PIC 99.  
        10 FILLER OCCURS 7 TIMES INDEXED BY WEEKW.  
            15 FILLER PIC 9.  
            15 FILLER PIC S99999V99.
```

FD LAST-YR-FILE LABEL RECORDS STANDARD.
01 LAST-YR-RECORD.
05 LAST-YR-WEEK-TABLE OCCURS 52 TIMES
INDEXED BY WEEKY WEEKZ.
10 LAST-YR-WEEK-NO PIC 99.
10 LAST-YR-AREA-TOTALS OCCURS 7 TIMES INDEXED BY AREAY.
15 LAST-YR-AREA-NO PIC 9.
15 LAST-YR-AREA-FIGURES PIC S99999V99.

FD PRINT-FILE LABEL RECORDS OMITTED.
01 PRINT-RECORD.
05 FILLER PIC X.
05 PRINT-DATA PIC X(95).
FD INQUIRY-FILE LABEL RECORDS OMITTED.
01 INQUIRY-RECORD PIC 99.

WORKING-STORAGE SECTION.
77 THIS-WEEK PIC 99 COMP-3.
77 LAST-WEEK PIC 99 COMP-3.
77 ANY-INQ PIC 9 VALUE 0.
77 INQ-COND-MET PIC 9.
77 NO-INQ-MSG PIC X(21) VALUE " NO INQUIRY WAS MADE."

01 NONE-MET-COND.
05 FILLER PIC X(17) VALUE " NO AREA IN WEEK ".
05 CON-MSG-WK PIC Z9.
05 FILLER PIC X(29) VALUE " EXCEEDED SALES OF LAST YEAR."

01 YTD-WORK-TABLE.
05 WEEK-TABLE OCCURS 1 TO 52 TIMES
DEPENDING ON THIS-WEEK
INDEXED BY WEEKT WEEKX.
10 WEEK-NO PIC 99.
10 AREA-TOTALS OCCURS 7 TIMES
INDEXED BY AREAT AREAX.
15 AREA-NO PIC 9.
15 AREA-FIGURES PIC S99999V99.

```

01 CURRENT-REPORT-PRINT.
05 FILLER PIC X (10) VALUE " WEEK NO.".
05 PRINT-WEEK PIC Z9.
05 PRINT-AREA VALUE " AREA ".
10 FILLER PIC X(4).
10 PRINT-AREA-NO PIC 9.
10 FILLER PIC X(4).
05 PRINT-AREA-FIGURES.
10 FILLER PIC X(17) VALUE "THIS WEEK SALES ".
10 PRINT-AREA-SALES PIC $$$,$$.99.
10 FILLER PIC X(5).
05 PRINT-AREA-LAST-WEEK.
10 FILLER PIC X(17) VALUE "LAST WEEK SALES ".
10 LAST-WEEK-SALES PIC $$$,$$.99.
10 FILLER PIC XX.
05 WEEK-PERCENT.
10 FILLER PIC X(15) VALUE "% CHANGE ".
10 WEEK-CHANGE PIC ++9.99.
10 FILLER PIC X(5).
05 PRINT-AREA-LAST-YEAR.
10 FILLER PIC X(17) VALUE "LAST YEAR SALES ".
10 LAST-YEAR-SALES PIC $$$,$$.99.
10 FILLER PIC XX.
05 YEAR-PERCENT.
10 FILLER PIC X(15) VALUE "% CHANGE ".
10 YEAR-CHANGE PIC ++9.99.

01 INQUIRY-PRINT-DATA.
05 FILLER PIC X(6) VALUE " WEEK ".
05 INQ-WK-NO PIC Z9.
05 FILLER PIC X(6) VALUE " AREA ".
05 INQ-AREA PIC 9.
05 FILLER PIC X(17) VALUE " $ INCREASE ".
05 INQUIRY-PRINT-FIGURE PIC $$.99.

01 LAST-WEEK-COMPARE.
05 FILLER PIC 99.
05 LW-AREA-TABLE OCCURS 7 TIMES INDEXED BY LWX.
10 LW-AREA-NO PIC 9.
10 LW-AREA-FIGURES PIC $99999V99.

01 ERROR-MSG.
05 FILLER PIC X.
05 ERROR-WEEK-NO PIC Z9.
05 ERROR-TEXT PIC X(40) VALUE
"EXCEEDS THIS-WEEK NO. INQUIRY IGNORED.".

```

PROCEDURE DIVISION.
HOUSEKEEPING-ROUTINE.

OPEN INPUT CURRENT-FILE LAST-YR-FILE INQUIRY-FILE,
OUTPUT PRINT-FILE.
READ LAST-YR-FILE AT END GO TO CLOSE-ROUTINE.
READ CURRENT-FILE AT END GO TO CLOSE-ROUTINE.
MOVE CURR-WEEK TO PRINT-WEEK.
WRITE PRINT-RECORD FROM CURRENT-REPORT-PRINT
AFTER ADVANCING 0.
MOVE SPACES TO CURRENT-REPORT-PRINT.
MOVE CURR-WEEK TO THIS-WEEK.
SET WEEKT WEEKX WEEKY TO THIS-WEEK.
IF THIS-WEEK EQUAL 1 GO TO FIRST-WEEK-ROUTINE.

LATER-WEEK-ROUTINE.

OPEN I-O YR-TO-DATE-FILE.
COMPUTE LAST-WEEK = THIS-WEEK - 1.
READ YR-TO-DATE-FILE INTO YTD-WORK-TABLE
AT END GO TO CLOSE-ROUTINE.
MOVE WEEK-TABLE (WEEKX - 1) TO LAST-WEEK-COMPARE.

WEEK-CALCULATION-ROUTINE.

MOVE CURRENT-SALE-RECORD TO WEEK-TABLE (WEEKT).
SET LWX AREAT AREAY TO 1.
PERFORM WEEK-PRINT THRU WEEK-PRINT-END 7 TIMES.

WRITE-FILE-ROUTINE.

REWRITE YR-TO-DATE-RECORD FROM YTD-WORK-TABLE.

INQUIRY-FIND-ROUTINE.

READ INQUIRY-FILE AT END GO TO CLOSE-ROUTINE.
MOVE 1 TO ANY-INQ.
IF INQUIRY-RECORD > THIS-WEEK MOVE INQUIRY-RECORD
TO ERROR-WEEK-NO,
WRITE PRINT-RECORD FROM ERROR-MSG AFTER ADVANCING 2,
GO TO INQUIRY-FIND-ROUTINE.
SET WEEKT WEEKY TO INQUIRY-RECORD.
SET AREAT AREAY TO 1.
MOVE INQUIRY-RECORD TO INQ-WK-NO.
MOVE 0 TO INQ-COND-MET.
PERFORM INQUIRY-PRINT THRU INQUIRY-PRINT-END UNTIL
AREAT = 8.
IF INQ-COND-MET EQUAL 0, MOVE INQUIRY-RECORD TO COND-MSG-WK,
WRITE PRINT-RECORD FROM NONE-MET-COND
AFTER ADVANCING 2.
GO TO INQUIRY-FIND-ROUTINE.

```

CLOSE-ROUTINE.
  IF ANY-INQ EQUAL 0, WRITE PRINT-RECORD FROM NO-INQ-MSG
    AFTER ADVANCING 2.
  CLOSE CURRENT-FILE YR-TO-DATE-FILE LAST-YR-FILE
    INQUIRY-FILE PRINT-FILE.
END-ROUTINE.
  STOP RUN.
FIRST-WEEK-ROUTINE.
  OPEN OUTPUT YR-TO-DATE-FILE.
  SET WEEKZ TO 52.
  MOVE LAST-YR-WEEK-TABLE (WEEKZ) TO LAST-WEEK-COMPARE.
  PERFORM WEEK-CALCULATION-ROUTINE.
  WRITE YR-TO-DATE-RECORD FROM YTD-WORK-TABLE.
  GO TO INQUIRY-FIND-ROUTINE.

WEEK-PRINT.
  COMPUTE WEEK-CHANGE = (AREA-FIGURES (WEEKT AREAT) -
    LW-AREA-FIGURES (LWX)) / LW-AREA-FIGURES (LWX) *
    100.00.
  COMPUTE YEAR-CHANGE = (AREA-FIGURES (WEEKT AREAT) -
    LAST-YR-AREA-FIGURES (WEEKY AREAY)) /
    LAST-YR-AREA-FIGURES (WEEKY AREAY) * 100.00.
  MOVE AREA-NO (WEEKT AREAT) TO PRINT-AREA-NO.
  MOVE AREA-FIGURES (WEEKT AREAT) TO PRINT-AREA-SALES.
  MOVE LW-AREA-FIGURES (LWX) TO LAST-WEEK-SALES.
  MOVE LAST-YR-AREA-FIGURES (WEEKY AREAY) TO LAST-YEAR-SALES.
  WRITE PRINT-RECORD FROM CURRENT-REPORT-PRINT
    AFTER ADVANCING 1.
  SET LWX AREAT AREAY UP BY 1.
WEEK-PRINT-END. EXIT.

INQUIRY-PRINT.
  SEARCH AREA-TOTALS VARYING AREAY
    AT END SET AREAT TO 8 GO TO INQUIRY-PRINT-END
  WHEN AREA-FIGURES (WEEKT AREAT) >
    LAST-YR-AREA-FIGURES (WEEKY AREAY)
    MOVE 1 TO INQ-COND-MET,
    COMPUTE INQUIRY-PRINT-FIGURE =
      AREA-FIGURES (WEEKT AREAT) -
      LAST-YR-AREA-FIGURES (WEEKY AREAY).
  MOVE AREA-NO (WEEKT AREAT) TO INQ-AREA.
  WRITE PRINT-RECORD FROM INQUIRY-PRINT-DATA
    AFTER ADVANCING 2.
  SET AREAT AREAY UP BY 1.
INQUIRY-PRINT-END. EXIT.

```

Chapter 13. Sort-Merge

Contents

About This Chapter	13-3
Introduction	13-4
Relationship with File Input and Output	13-4
Environment Division in the Sort-Merge Module—Input-Output Section	13-4
FILE-CONTROL Paragraph	13-5
Function	13-5
General Format	13-5
FILE-CONTROL Entry	13-6
Function	13-6
General Format	13-6
Syntax Rules	13-6
General Rules	13-6
I-O-CONTROL Paragraph	13-8
Function	13-8
General Format	13-8
Syntax Rules	13-8
General Rules	13-9
Data Division in the Sort-Merge Module	13-10
Sort-Merge File Description - Complete Entry Skeleton	13-10
General Format	13-10
Syntax Rules	13-11
DATA RECORDS Clause	13-12
Function	13-12
General Format	13-12
Syntax Rule	13-12
General Rules	13-12
RECORD CONTAINS Clause	13-13
Function	13-13
General Format	13-13
General Rule	13-13
Procedure Division in the Sort-Merge Module	13-14
MERGE Statement	13-14
Function	13-14
General Format	13-14
Syntax Rules	13-14
General Rules	13-15
RELEASE Statement	13-18
Function	13-18
General Format	13-18
Syntax Rules	13-18
General Rules	13-18
RETURN Statement	13-19
Function	13-19
General Format	13-19
Syntax Rules	13-19
General Rules	13-19
SORT Statement	13-21
Function	13-21
General Format	13-21
Syntax Rules	13-21
General Rules	13-22
Sort-Merge Sample Program	13-26

About This Chapter

This chapter describes the AIX VS COBOL Sort-Merge module and how it provides the capability to order one or more files of records, or to combine two or more identically ordered files of records, according to a set of user-defined keys contained within each record.

Introduction

The Sort-Merge module provides the capability to order one or more files of records, or to combine two or more identically ordered files of records, according to a set of user-specified keys contained within each record. Optionally, a user may apply some special processing to each of the individual records by input or output procedures. This special processing may be applied before and/or after the records are ordered by the SORT statement or after the records have been combined by the MERGE statement.

Relationship with File Input and Output

The files specified in the USING and GIVING phrases of the SORT and MERGE statements must be described implicitly or explicitly in the FILE CONTROL paragraph as having sequential, *relative or indexed* organization and sequential access mode. No input-output statement may be executed for the file named in the sort-merge file description. OSVS VSC2

Environment Division in the Sort-Merge Module—Input-Output Section

The Environment Division in the sort-merge module may contain a FILE-CONTROL paragraph, a file-control entry, and an I-O-CONTROL paragraph.

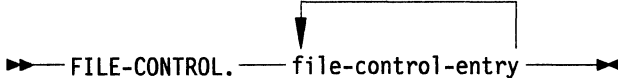
FILE-CONTROL Paragraph

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

General Format

The following figure shows the format of the FILE-CONTROL paragraph:



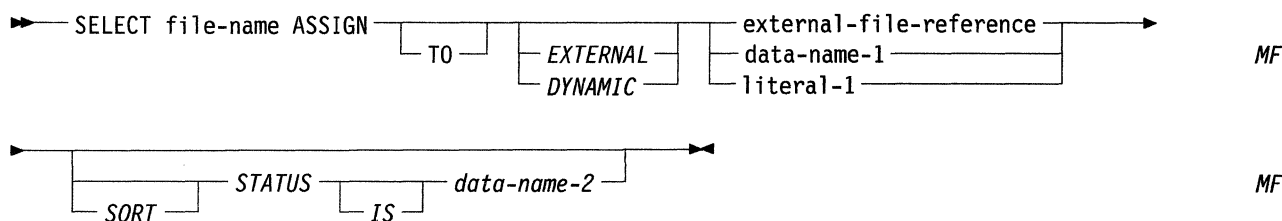
FILE-CONTROL Entry

Function

The FILE-CONTROL entry names a sort or merge file and assigns the file to a memory medium.

General Format

The following figure shows the FILE-CONTROL entry:



Syntax Rules

The following rules apply to the FILE-CONTROL paragraph:

1. Each sort or merge file described in the Data Division must be named only once as file name in the FILE-CONTROL paragraph. Each sort or merge file specified in the file control entry must have a sort-merge file description entry in the Data Division.
2. Only the ASSIGN clause is permitted to follow a file name representing a sort-merge file in the FILE-CONTROL paragraph.

General Rules

The following rules apply to the FILE-CONTROL paragraph:

1. The ASSIGN clause assigns the sort or merge file referenced by file name to a memory medium.
2. *When the SORT STATUS clause is specified, a value is placed into the two-character data-item specified by data-name-2 after the execution of each sort operation. This value indicates the status at completion of the operation.* MF

Valid combinations of status keys 1 and 2 indicate the status of the sort operation. Refer to "I-O Status" on page 8-8 for explanations of status keys 1 and 2 and definitions of status.

The following combinations of status keys are possible:

Table 13-1. Status Key Combinations		
Status Key 1	Status Key 2	Status
0	0	Successful completion.
3	0	Permanent error.
9		Operating system error message number in status key 2.

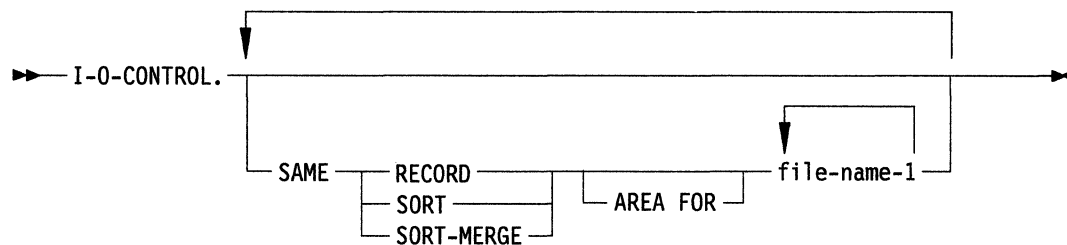
I-O-CONTROL Paragraph

Function

The I-O-CONTROL paragraph specifies the memory area that is to be shared by different files.

General Format

The following figure shows the format of the I-O-CONTROL paragraph:



Syntax Rules

The following syntax rules apply to the I-O-CONTROL paragraph:

1. The I-O-CONTROL paragraph is optional.
2. In the SAME AREA clause, SORT and SORT-MERGE are equivalent.
3. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause.
4. The three formats of the SAME clause (SAME RECORD AREA, SAME SORT AREA, SAME SORT-MERGE AREA) are considered separately in the following.

More than one SAME clause may be included in a program. However:

- a. A file name must not appear in more than one SAME RECORD AREA clause.
 - b. A file name for a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.
 - c. If a file name not representing a sort or merge file appears in a SAME AREA clause and one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all of the files named in the SAME AREA clause must be named in the SAME SORT AREA or SAME SORT-MERGE AREA clause(s).
5. The files referenced in the SAME SORT AREA, SAME SORT-MERGE AREA, or SAME RECORD AREA clause need not all have the same organization or access.

General Rules

The following rules apply to the I-O-CONTROL paragraph:

1. The SAME RECORD AREA clause specifies that two or more files will use the same memory area for processing the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered a logical record of each opened output file listed in the SAME RECORD AREA clause and of the most recently read input file listed in the SAME RECORD AREA clause. This is implicit redefinition of the area. Records are aligned on the leftmost character position.
2. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file names must represent a sort or merge file. Other files may also be named in the clause. This clause specifies that memory is shared as follows:
 - a. The SAME SORT AREA or SAME SORT-MERGE AREA clause specifies memory area made available for sorting or merging each sort or merge file named. Thus any memory area allocated for the sorting or merging may be used or reused by any of the other sort or merge files.
 - b. In addition, memory areas assigned to files that are not sort or merge files may be allocated as needed for sorting or merging the files named in the SAME SORT AREA or SAME SORT-MERGE AREA clause.
 - c. Files other than sort or merge files do not share the same memory area with each other. To specify that these files should share the same memory area, include a SAME AREA or SAME RECORD AREA clause naming these files in the program.
 - d. Nonsort-merge files named in the SAME AREA or SAME RECORD AREA clause must not be open during the execution of a SORT or MERGE statement using a sort or merge file named in the clause.

Data Division in the Sort-Merge Module

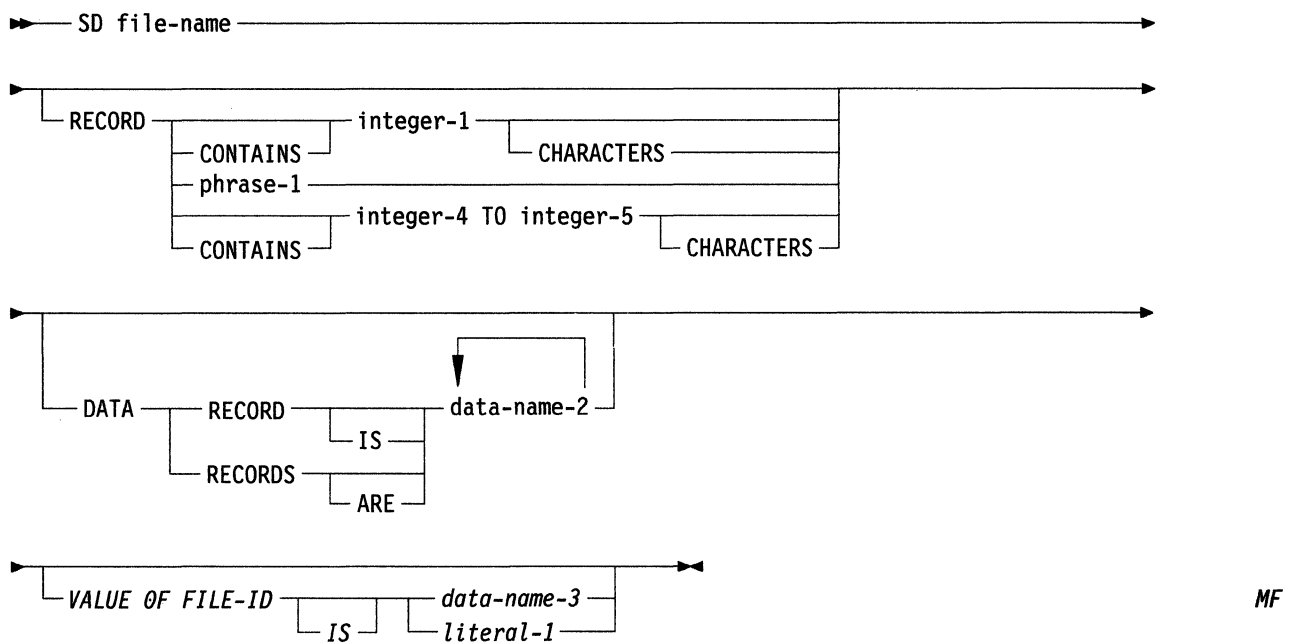
An SD file description describes the size and names of the data records associated with the file to be sorted or merged. None of the label procedures are user controlled. The rules for blocking and internal memory depend on the SORT and MERGE statements.

Sort-Merge File Description - Complete Entry Skeleton

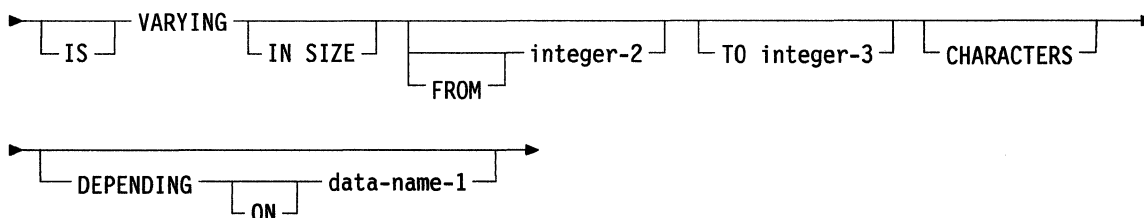
The sort-merge file description furnishes information concerning the physical structure, identification, and record names of the file to be sorted or merged.

General Format

The following figure shows the sort-merge file description:



where phrase-1 is:



Syntax Rules

The following syntax rules apply to the sort-merge file description:

1. The level indicator SD identifies the beginning of the sort-merge file description. The level indicator SD must precede the file name.
2. The clauses that follow the name of the file are optional and their order of appearance is immaterial. They are for documentation purposes only.
3. One or more record description entries must follow the sort-merge file description entry. However, no input-output statements may be executed for this file.
4. *If the VALUE OF FILE-ID clause is specified, literal-1 must be a nonnumeric literal and cannot be a figurative constant.* MF

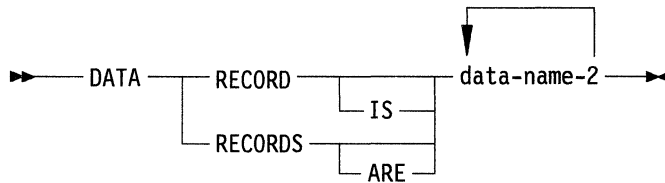
DATA RECORDS Clause

Function

The DATA RECORDS clause documents the names of data records and their associated file.

General Format

The following figure shows the format of the DATA RECORDS clause:



Syntax Rule

The listed data-name-2 items are data records that must have 01 level number record descriptions with the same names.

General Rules

The following rules apply to the DATA RECORDS clause:

1. The presence of more than one data name indicates that the file contains more than one type of data record. These records may have differences, such as different sizes or different formats. The order in which they are listed is not significant.
2. Conceptually, all data records within a file share the same area. This is not altered by the presence of more than one type of data record within the file.

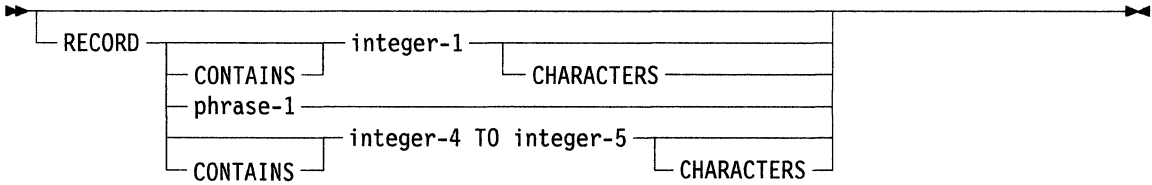
RECORD CONTAINS Clause

Function

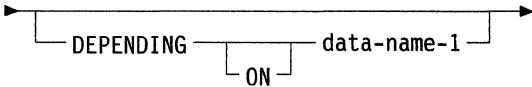
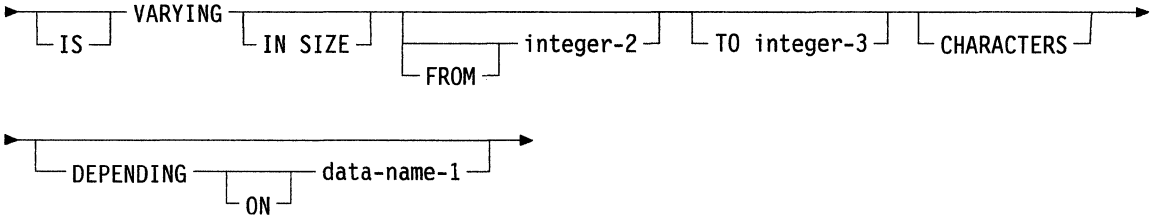
The RECORD CONTAINS clause specifies the size of data records.

General Format

The following figure shows the format of the RECORDS CONTAIN clause:



where phrase-1 is:



General Rule

The RECORD clause for the Sort-Merge module is identical to the RECORD clause for the sequential I-O module. Refer to "RECORD Clause" on page 8-47 for a full specification of this clause.

Procedure Division in the Sort-Merge Module

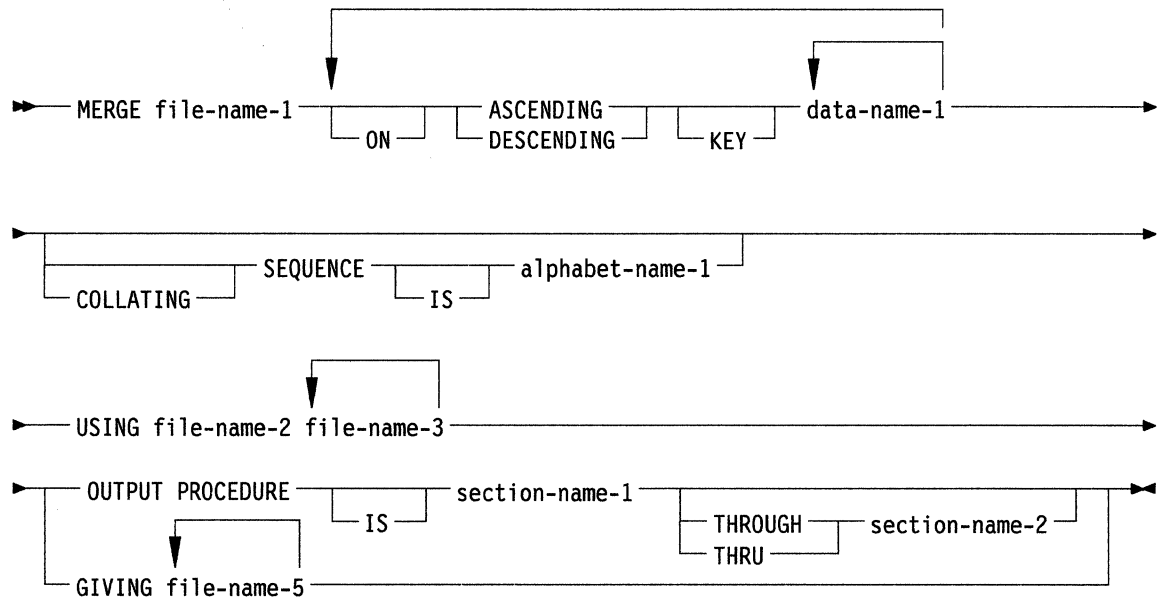
MERGE Statement

Function

The MERGE statement combines two or more identically sequenced files on a set of specified keys and makes records available, in merge order, to an output procedure or file.

General Format

The following figure shows the format of the MERGE statement:



Syntax Rules

The following syntax rules apply to the MERGE statement:

1. file-name-1 must be described in a sort-merge file description entry in the Data Division.
2. section-name-1 represents the name of an output procedure.
3. If the file referenced by file-name-1 contains variable-length records, the size of records contained in file-name-2 and file-name-3 must not be less than the smallest record, nor greater than the largest record described for file-name-1. If the file referenced by file-name-1 contains fixed-length records, the size of records contained in the file refer-

enced by file-name-2 and file-name-3 must not be greater than the largest record described for file-name-1.

4. file-name-2, file-name-3, and file-name-5 must be described in a file description entry in the Data Division, not in a sort-merge file description entry.
5. The words THRU and THROUGH are equivalent.
6. data-name-1 is a KEY data name and is subject to the following rules:
 - a. The data-items identified by KEY data names must be described in records associated with file-name-1.
 - b. KEY data names may be qualified.
 - c. The data-items identified by KEY data names must not be variable-length items.
 - d. If file-name-1 has more than one record description, then the data-items identified by KEY data names need be described in only one of the record descriptions.
 - e. None of the data-items identified by KEY data names can be described by an entry which contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.
7. No more than one file name from a multiple file reel can appear in the MERGE statement.
8. File names must not be repeated within the MERGE statement.
9. MERGE statements may appear anywhere except the declaratives portion of the Procedure Division.
10. If file-name-5 references an indexed file, the first specification of data-name-1 must be associated with an ASCENDING phrase. The data-item referenced by data-name-1 must occupy the same character positions in its record as the data-item associated with the prime record key for the file.
11. If the GIVING phrase is specified and the file referenced by file-name-5 contains variable-length records, the record size of the file referenced by file-name-1 must not be less than the smallest record nor greater than the largest record described for file-name-5. If the file referenced by file-name-5 contains fixed-length records, the record size of the file referenced by file-name-1 must not be greater than the largest record described for file-name-5.

General Rules

The following rules apply to the MERGE statement:

1. The MERGE statement merges all records contained on file-name-2 and file-name-3.
2. The data names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance, regardless of their division into KEY phrases.
 - a. When the ASCENDING phrase is specified, the merged sequence begins at the lowest value of the contents of the data-items identified by the KEY data names and proceeds to the highest value, according to the rules for comparison of operands in a relation condition.
 - b. When the DESCENDING phrase is specified, the merged sequence begins at the highest value of the contents of the data-items identified by the KEY data names and proceeds to the lowest value, according to the rule for comparison of operands in a relation condition.
3. The collating sequence for comparison of the nonnumeric key data-items specified is determined in the following order of precedence:
 - a. The collating sequence established by the COLLATING SEQUENCE phrase in that MERGE statement

-
- b. The collating sequence established as the program collating sequence.
 4. The output procedure must consist of one or more contiguous sections in a source program. The sections may not form part of any other procedure. In order to make merged records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the returned records from file-name-1 one at a time in merge order. The restrictions on the procedural statements within the output procedure are as follows:
 - a. The output procedure must not contain any transfers of control to points outside the output procedure. ALTER, GO TO and PERFORM statements in the output procedure are not permitted to refer to procedure names outside the output procedure. COBOL statements implying transfer of control to declaratives are allowed.
 - b. The output procedures must not contain any SORT or MERGE statements.
 - c. The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedures. ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure names within the output procedures.
 5. If an output procedure is specified, control passes to it during execution of the MERGE statement. The IBM AIX VS COBOL system inserts a return mechanism at the end of the last section in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the merge, and then passes control to the next executable procedure. The merge procedure has then reached 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.
 6. Segmentation, as defined in Chapter 16, "Segmentation," can be applied to programs containing the MERGE statement. However, the following restrictions apply:
 - a. If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by the MERGE statement must appear:
 - Totally within nonindependent segments, or
 - Contained in a single independent segment.
 - b. If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:
 - Totally within nonindependent segments, or
 - Contained in the same independent segment as the MERGE statement.
 7. If the GIVING phrase is specified, all the merged records are written on file-name-5 as the implied output procedure for the MERGE statement. At the start of the execution of the MERGE statement, the file referenced by file-name-5 must not be in the open mode. For each of the files referenced by file-name-5, the execution of the MERGE statement causes the following actions:
 - a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the output phrase had been executed.
 - b. The merged logical records are returned and written onto the file. Each record is written as if a WRITE statement without any optional phrases had been executed.

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 MERGE statement, the content of the relative key data-item indicates the last record returned to the file.
 - c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

These implicit functions are performed such that any associated USE 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-5. 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 the USE procedure or if no such USE procedure is specified, the processing of the file is terminated as in paragraph 7c on page 13-16.

8. In the case of a match between the contents of the data-items identified by all the KEY data names between records from two or more input files (file-name-2, file-name-3, ...), the records are written on file-name-5 or returned to the output procedure in the same order as the input files specified in the MERGE statement. Equal compare is determined according to the rules for comparison of operands in a relation condition.
9. The results of the merge operation are predictable only when the records in the files referenced by file-name-2, file-name-3, ..., are ordered as described in the ASCENDING or DESCENDING KEY clause associated with the MERGE statement.
10. If the file referenced by file-name-1 contains only fixed-length records, any record in the file referenced by file-name-2 or file-name-3 containing fewer character positions than the fixed-length is space-filled on the right, when the record is released to the file referenced by file-name-1.
11. If the file referenced by file-name-5 contains only fixed-length records, any record in the file referenced by file-name-1 containing fewer character positions than the fixed-length is space-filled on the right, when that record is returned to the file referenced by file-name-5.

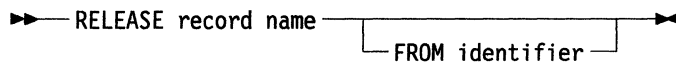
RELEASE Statement

Function

The **RELEASE** statement transfers records to the initial phrase of a **SORT** operation.

General Format

The following figure shows the format of the **RELEASE** statement:



Syntax Rules

The following syntax rules apply to the **RELEASE** statement:

1. A **RELEASE** statement may only be used within the range of an input procedure associated with a **SORT** statement for a file whose sort-merge file description entry contains record-name. Refer to “**SORT Statement**” on page 13-21.
2. Record-name must be the name of a logical record in the associated sort-merge file description entry and may be qualified.
3. Record-name and identifier must not refer to the same memory area.

General Rules

The following rules apply to the **RELEASE** statement:

1. The execution of a **RELEASE** statement releases the record named by record-name to the initial phase of a sort operation.
2. If the **FROM** phrase is used, the contents of the identifier data area are moved to record-name, then the contents of record-name are released to the sort file. Moving files takes place according to the rules specified for the **MOVE** statement without the **CORRESPONDING** phrase. The information in the record area is no longer available, but the information in the data area associated with identifier is available.
3. After the execution of the **RELEASE** statement, the logical record is no longer available in the record area unless the associated sort-merge file is named in a **SAME RECORD AREA** clause. The logical record is also available to the program as a record of other files referenced in the same **SAME RECORD AREA** clause as the associated sort-merge file and the file associated with record-name. When control passes from the input procedure, the file consists of all those records which were placed in it by the execution of **RELEASE** statements.

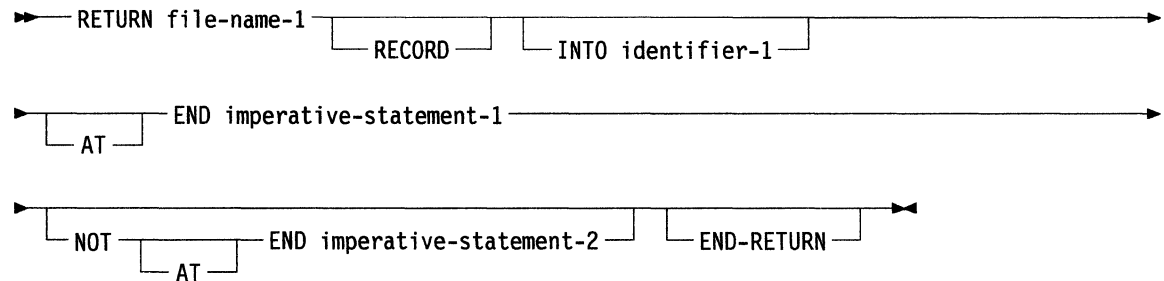
RETURN Statement

Function

The RETURN statement obtains either sorted records from the final phase of a SORT operation or merged records during a MERGE operation.

General Format

The following figure shows the format of the RETURN statement:



Syntax Rules

The following syntax rules apply to the RETURN statement:

1. file-name-1 must be described by a sort-merge file description entry in the Data Division.
2. A RETURN statement may only be used within the range of an output procedure associated with a SORT or MERGE statement for file-name-1.
3. The INTO phrase must not be used when the input file contains logical records of variable size as indicated by their record descriptions. The memory area associated with identifier and the record area associated with file-name-1 must not be the same memory area.

General Rules

The following rules apply to the RETURN statement:

1. When the logical records of a file are described with more than one record description, these records automatically share the same memory area. This is equivalent to an implicit redefinition of the area. The contents of any data-items that lie beyond the range of the current data record are undefined at the completion of the execution of the RETURN statement.

-
2. The execution of the RETURN statement makes the next existing record in the file available in the record area associated with file referenced, as determined by the keys listed in the SORT or MERGE statement. If no next logical record exists in the file referenced, the AT END condition exists and control is transferred to imperative-statement-1 of the AT END phrase. Execution continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that transfers explicit control is executed, control is transferred according to the rules for that statement. Otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the RETURN statement. The NOT AT END phrase is ignored. When the AT END condition occurs, execution of the RETURN statement is unsuccessful and the contents of the record area associated with file name are undefined. After the execution of imperative-statement-1 in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.
 3. If an AT END condition does not occur during the execution of a RETURN statement, then after the record is available and after executing any implicit move resulting from an INTO phrase, control is transferred to imperative-statement-2. If imperative-statement-2 is not specified, control is transferred to the end of the RETURN statement.
 4. The END-RETURN phrase delimits the scope of the RETURN statement.
 5. The INTO phrase may be specified in a RETURN statement:
 - a. If only one record description is subordinate to the sort-merge file description entry, or
 - b. If all record-names associated with file-name-1 and the data-item referenced by identifier-1 describe a group item or an elementary alphanumeric item.
 6. The result of the execution of a RETURN statement with the INTO phrase is equivalent to the application of the following rules in the order specified:
 - a. The execution of the same RETURN statement without the INTO phrase.
 - b. The current record is moved from the record area to the area specified by identifier-1 according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is determined by rules specified for the RECORD clause. If the file description entry contains a RECORD IS VARYING clause, the implied move is a group move. The implied MOVE statement does not occur if the execution of the RETURN statement was unsuccessful. Any subscripting 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.

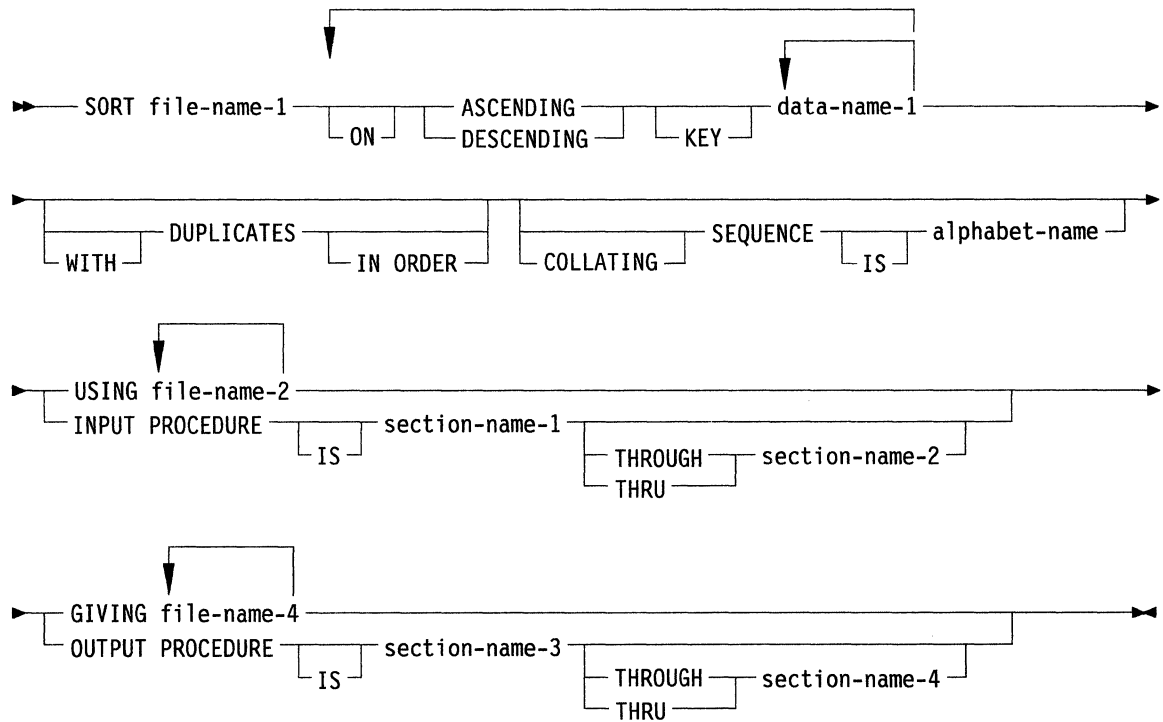
SORT Statement

Function

The SORT statement creates a sort file by executing input procedures or by transferring records from another file. It then sorts the records in the sort file on a set of specified keys. In the final phase of the sort operation, it makes each record from the sort file available to output procedures or an output file, in sorted order.

General Format

The following figure shows the format of the SORT statement:



Syntax Rules

The following syntax rules apply to the SORT statement:

1. file-name-1 must be described in a sort-merge file description entry in the Data Division.
2. section-name-1 represents the name of an input procedure. section-name-3 represents the name of an output procedure.

-
3. file-name-2 and file-name-4 must be described in a file description entry in the Data Division, not in a sort-merge file description entry. The actual size of the logical record(s) described for file-name-2 and file-name-4 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items making up these records are not identical, the programmer must describe the corresponding records so equal amounts of character positions are allocated for the corresponding records.
 4. data-name-1 represents a KEY data name, subject to the following rules:
 - a. The data-items identified by KEY data names must be described in records associated with file-name-1.
 - b. KEY data names may be qualified.
 - c. The data-items identified by KEY data names must not be variable in length items.
 - d. If file-name-1 has more than one record description, then the data items identified by KEY data names need be described in only one of the record descriptions.
 - e. None of the data-items identified by KEY data names can be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.
 5. If the USING phrase is specified and the file referenced by file-name-1 contains variable-length records, the size of the records contained in the file referenced by file-name-2 must not be less than the smallest record nor larger than the largest record described for file-name-1. If the file referenced by file-name-1 contains fixed-length records, the size of the records contained in the file referenced by file-name-2 must not be larger than the largest record described for the file referenced by file-name-1.
 6. The words THRU and THROUGH are equivalent.
 7. SORT statements may appear anywhere except in the declaratives portion of the Procedure Division or in an input or output procedure associated with a SORT or MERGE statement.
 8. If file-name-4 references an indexed file, the first specification of data-name-1 must occupy the same character positions in its record as the data-item associated with the prime record key for that file.
 9. If the GIVING phrase is specified and file-name-4 contains variable-length records, the size of the records in file-name-1 must not be less than the smallest record nor greater than the largest record described for file-name-4. If the file referenced by file-name-4 contains fixed-length records, the size of the records in the file referenced by file-name-1 must not be greater than the largest record described for file-name-4.
 10. The file referenced by file-name-2 may reside on the same multiple file reel.

General Rules

The following rules apply to the SORT statement:

1. If the file referenced by file-name-1 contains only fixed-length records, any record in the file referenced by file-name-2 containing fewer character positions than the fixed-length is space-filled on the right. Space-filling begins after the last character in the record, when the record is released to the file referenced by file-name-1.
2. The data names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance regardless of how they are divided into KEY phrases.
 - a. When the ASCENDING phrase is specified, the sorted sequence moves from the lowest value of the contents of the data-items identified by the KEY data names to the highest value, according to the rules for comparison of operands in a relation condition.

-
- b. When the DESCENDING phrase is specified, the sorted sequence is from the highest value of the contents of the data items identified by the KEY data names to the lowest value, according to the rules for comparison of operands in a relation condition.
 3. If the DUPLICATES phrase is not specified and the contents of all the key data-items for one data record are equal to the contents of the corresponding key data-items for one or more other data records, then the record's return order is undefined.
 4. If the DUPLICATES phrase is specified and the contents of all the key data-items for one data record are equal to the contents of the corresponding key data-items for one or more other data records, then the order of return of these records is:
 - a. The order of the associated input files as specified in the SORT statement, or by means of a run-time switch. Refer to the *User's Guide* for details of run-time switches. Within a given input file the order is that in which the records are accessed.
 - b. The order in which these records are released by an input procedure, when an input procedure is specified.
 5. The collating sequence that applies to the comparison of the nonnumeric key data-items specified is determined in the following order of precedence:
 - a. First, the collating sequence established by the COLLATING SEQUENCE phrase in the SORT statement.
 - b. Second, the collating sequence established as the program collating sequence.
 6. The execution of the SORT statement consists of three distinct phases as follows:
 - a. Records are made available to the file referenced by file-name-1. This is achieved either by the execution of RELEASE statements in the input procedure or by the implicit execution of READ statements for file-name-2. When this phase commences, the file referenced by file-name-2 must not be in the open mode. When this phase terminates, the file referenced by file-name-2 is not in the open mode.
 - b. The file referenced by file-name-1 is sequenced. No processing of the files referenced by file-name-2 and file-name-4 takes place during this phase.
 - c. The records of the file referenced by file-name-1 are made available in sorted order. The sorted records are either written to the file referenced by file-name-4 or made available to the output procedure by the execution of a RETURN statement. When this phase commences, the file referenced by file-name-4 must not be in the open mode. When this phase terminates, the file referenced by file-name-4 is not in the open mode.
 7. The input procedure may consist of any procedure needed to select, modify, or copy the records to file-name-1 one at a time by the RELEASE statement. The range includes all statements executed as the result of a transfer of control by CALL, GO TO, and PERFORM statements in the range of the input procedure. The range also includes all statements in declarative procedures 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.
 8. If an input procedure is specified, control is passed to the input procedure before the file referenced by file-name-1 is sorted by the SORT statement. The AIX VS COBOL system inserts a return mechanism at the end of the last section in the input procedure. When control passes the last statement in the input procedure, the records released to file-name-1 are sorted.
 9. If the USING phrase is specified, all the records in the file(s) referenced by file-name-2 are transferred to the file referenced by file-name-1. For each of the files referenced by file-name-2, the execution of the SORT statement causes the following actions:
 - a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase had been executed.

-
- b. The logical records are obtained and released to the sort operation. Each record is obtained as if a READ statement with the NEXT and the AT END phrases had been executed.

For a relative file, the content of the relative key data-item is defined after the execution of the SORT statement, if file-name-2 is referenced in the GIVING phrase.

- c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. This termination is performed before the file referenced by file-name-1 is sorted by the SORT statement.

Any associated USE AFTER EXCEPTION/ERROR procedures are executed when these implicit functions are performed. 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 for file-name-2.

10. The output procedure may consist of any procedure needed to select, modify, or copy the records from file-name-1, one at a time by the RETURN statement in sorted order. The range includes all statements executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the output procedure. It also includes all statements in declarative procedures 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.
11. If an output procedure is specified, control passes to it after file-name-1 has been sorted by the SORT statement. The AIX VS COBOL system inserts a return mechanism at the end of the last section in the output procedure. When control passes the last statement in the output procedure, the return mechanism terminates the sort. The return mechanism then passes control to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point where it can request the next record in sorted order. The RETURN statements in the output procedure are the requests for the next record.
12. If file-name-4 contains only fixed-length records, any record in file-name-1 containing fewer character positions than the fixed-length is space-filled to the right, beginning after the last character in the record, when that record is returned to the file referenced by file-name-4.
13. If the GIVING phrase is specified, all the sorted records are written on the file referenced by file-name-4 as the implied output procedure for the SORT statement. For each of the files referenced by file-name-4, the execution of the SORT statement causes the following actions:
 - a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed. This initiation is performed after the execution of any input procedure.
 - b. The sorted logical records are returned and written onto the file. The records are 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.
 - c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed. However, the execution of the USE procedure must not cause the execution of any statement manipulating the file referenced by, or accessing the record area for file-name-4. 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 the USE procedure or if no such USE procedure is specified, the processing of the file is terminated as in 13c.

-
14. Segmentation, as defined in Chapter 16, "Segmentation," can be applied to programs containing the SORT statement. However, the following restrictions apply:
 - a. If a SORT statement appears in a section that is not in an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:
 - Totally in nonindependent segments, or
 - Contained in a single independent segment.
 - b. If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:
 - Totally in nonindependent segments, or
 - Within the same independent segment as that SORT statement.
 15. If the USING phrase is specified, all the records in file-name-2 are transferred automatically to file-name-1. At the times of execution of the SORT statement, file-name-2 must not be open. The SORT statement automatically initiates the processing of, makes available the logical records for, and terminates the processing of file-name-2. Any associated USE procedure is executed when these implicit functions are performed. The terminating function for all files is performed as if a CLOSE statement without optional phrases had been executed for each file. The SORT statement also automatically performs the implicit functions of moving the records from the file area of file-name-2 to the file area for file-name-1, and the release of records to the initial phase of the sort operation.

Sort-Merge Sample Program

The following example shows a sample sort-merge program:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. SORT-IT.  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER. IBM-RISC-6000.  
OBJECT-COMPUTER. IBM-RISC-6000.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT NET-FILE-IN ASSIGN TO "NETFILE.IN".  
    SELECT NET-FILE-OUT ASSIGN TO "NETFILE.OUT".  
    SELECT NET-FILE-WORK ASSIGN TO "NETFILE.WRK".  
    SELECT YTD-NET-FILE-IN ASSIGN TO "YTDNET.IN".  
    SELECT YTD-NET-FILE-MASTER  
        ASSIGN TO "YTDNET.MST".
```

```
DATA DIVISION.  
FILE SECTION.
```

```
SD NET-FILE-WORK  
  DATA RECORD IS SALES-RECORD.  
01 SALES-RECORD.  
   05 EMPL-NO           PICTURE 9(6).  
   05 DEPT              PICTURE 9(2).  
   05 NET-SALES         PICTURE 9(7)V99.  
   05 NAME-ADDR         PICTURE X(61).  
   05 MONTH             PICTURE X(2).
```

```
FD NET-FILE-IN  
  LABEL RECORDS ARE STANDARD  
  DATA RECORD IS NET-CARD-IN.  
01 NET-CARD-IN.  
   05 EMPL-NO-IN        PICTURE 9(6).  
   05 DEPT-IN           PICTURE 9(2).  
   88 OFF-SITE-LOCATION VALUES ARE 7, 9.  
   05 NET-SALES-IN      PICTURE 9(7)V99.  
   05 NAME-ADDR-IN      PICTURE X(61).  
   05 MONTH-IN          PICTURE X(2).
```

FD NET-FILE-OUT
LABEL RECORDS ARE STANDARD
DATA RECORD IS NET-CARD-OUT.
01 NET-CARD-OUT.
05 EMPL-NO-OUT PICTURE 9(6).
05 DEPT-OUT PICTURE 9(2).
05 NET-SALES-OUT PICTURE 9(7)V99.
05 NAME-ADDR-OUT PICTURE X(61).
05 MONTH-OUT PICTURE X(2).

FD YTD-NET-FILE-IN
LABEL RECORDS ARE STANDARD
DATA RECORD IS YTD-NET-CARD-IN.
01 YTD-NET-CARD-IN.
05 YTD-EMPL-NO PICTURE 9(6).
05 YTD-DEPT PICTURE 9(2).
05 YTD-NET-SALES PICTURE 9(7)V99.
05 YTD-NAME-ADDR PICTURE X(61).
05 YTD-MONTH PICTURE X(2).

FD YTD-NET-FILE-MASTER
LABEL RECORDS ARE STANDARD
DATA RECORD IS YTD-NET-CARD-MASTER.
01 YTD-NET-CARD-MASTER.
05 EMPL-NO-MASTER PICTURE 9(6).
05 DEPT-MASTER-MASTER PICTURE 9(2).
05 NET-SALES-MASTER-MASTER PICTURE 9(7)V99.
05 NAME-ADDR-MASTER-MASTER PICTURE X(61).
05 MONTH-MASTER-MASTER PICTURE X(2).

PROCEDURE DIVISION.
ELIM-DEPT-OFF-SITE-NO-PRINTOUT.
SORT NET-FILE-WORK
ASCENDING KEY DEPT
DESCENDING KEY NET-SALES
INPUT PROCEDURE SCREEN-DEPT
GIVING NET-FILE-OUT.

CHECK-RESULTS SECTION.

C-R-1.

OPEN INPUT NET-FILE-OUT.

C-R-2.

READ NET-FILE-OUT AT END GO TO C-R-FINAL.
DISPLAY EMPL-NO-OUT DEPT-OUT NET-SALES-OUT
NAME-ADDR-OUT.

C-R-3.

GO TO C-R-2.

C-R-FINAL.

CLOSE NET-FILE-OUT.

UPDATE-YEARLY-REPORT SECTION.

U-Y-R-1.

SORT NET-FILE-WORK
ASCENDING KEY DEPT
ASCENDING KEY EMPL-NO
USING NET-FILE-IN
GIVING NET-FILE-OUT.

U-Y-R-2.

MERGE NET-FILE-WORK
ASCENDING KEY DEPT
ASCENDING KEY EMPL-NO
ASCENDING KEY MONTH
USING YTD-NET-FILE-IN, NET-FILE-OUT
GIVING YTD-NET-FILE-MASTER.
STOP RUN.

SCREEN-DEPT SECTION.

S-D-1.

OPEN INPUT NET-FILE-IN.

S-D-2.

READ NET-FILE-IN AT END GO TO S-D-FINAL.
DISPLAY EMPL-NO-IN DEPT-IN NET-SALES-IN
NAME-ADDR-IN.

S-D-3.

IF OFF-SITE-LOCATION GO TO S-D-2
ELSE
MOVE NET-CARD-IN TO SALES-RECORD
RELEASE SALES-RECORD
GO TO S-D-2.

S-D-FINAL.

CLOSE NET-FILE-IN.

S-D-END.

EXIT.

Chapter 14. Report Writer

Contents

About This Chapter	14-5
Introduction	14-6
REPORT SECTION	14-6
Report Structure	14-6
Vertical Spacing	14-6
Horizontal Spacing	14-7
Data Manipulation	14-7
Report Subdivisions	14-7
Procedure Division Report Writer Statements	14-8
Language Concepts	14-8
Environment Division in the Report Writer Module	14-10
INPUT-OUTPUT Section	14-10
FILE-CONTROL Paragraph	14-10
Function	14-10
Additional Syntax Rule	14-10
I-O-CONTROL Paragraph	14-10
Function	14-10
Additional Syntax Rule	14-10
Data Division in the Report Writer Module	14-11
File Description Entry	14-11
Function	14-11
General Format	14-12
Syntax Rules	14-13
General Rules	14-13
REPORT Clause	14-14
Function	14-14
General Format	14-14
Syntax Rules	14-14
General Rules	14-14
Example	14-15
REPORT SECTION	14-16
Report Description Entry	14-16
Report Group Description Entry	14-16
REPORT Description Entry	14-17
CODE Clause	14-19
Function	14-19
General Format	14-19
Syntax Rules	14-19
General Rules	14-19
CONTROL Clause	14-20
Function	14-20
General Format	14-20
Syntax Rules	14-20
General Rules	14-20
PAGE Clause	14-22
Function	14-22
General Format	14-22
Syntax Rules	14-22
General Rules	14-23
Page Regions	14-25
Example 1	14-25
Example 2	14-26
Example 3	14-27
Report Group Description Entry	14-28
Function	14-28
General Format	14-28
Syntax Rules	14-31
Presentation Rules Tables	14-32
COLUMN NUMBER Clause	14-45

Function	14-45
General Format	14-45
Syntax Rules	14-45
Data-Name	14-46
Function	14-46
General Format	14-46
Syntax Rule	14-46
General Rules	14-46
GROUP INDICATE Clause	14-47
Function	14-47
General Format	14-47
Syntax Rule	14-47
General Rules	14-47
Level-Number	14-48
Function	14-48
General Format	14-48
Syntax Rules	14-48
General Rules	14-48
LINE NUMBER Clause	14-49
Function	14-49
General Format	14-49
Syntax Rules	14-49
General Rules	14-50
NEXT GROUP Clause	14-51
Function	14-51
Syntax Rules	14-51
General Rules	14-51
SIGN Clause	14-53
Function	14-53
General Format	14-53
Syntax Rules	14-53
General Rules	14-53
SOURCE Clause	14-55
Function	14-55
General Format	14-55
Syntax Rules	14-55
General Rule	14-55
SUM Clause	14-56
Function	14-56
General Format	14-56
Syntax Rules	14-56
General Rules	14-57
TYPE Clause	14-59
Function	14-59
General Format	14-59
Syntax Rules	14-59
General Rules	14-60
USAGE Clause	14-63
Function	14-63
General Format	14-63
Syntax Rules	14-63
General Rules	14-63
VALUE Clause	14-64
Function	14-64
Syntax Rules	14-64
General Rules	14-64
Example	14-65
Procedure Division in the Report Writer Module	14-66
CLOSE Statement	14-67
Additional Syntax Rule	14-67
Additional General Rule	14-67
GENERATE Statement	14-68

Function	14-68
General Format	14-68
Syntax Rules	14-68
General Rules	14-68
INITIATE Statement	14-71
Function	14-71
General Format	14-71
Syntax Rule	14-71
General Rules	14-71
OPEN Statement	14-72
Additional Syntax Rules	14-72
Additional General Rule	14-72
SUPPRESS Statement	14-73
Function	14-73
General Format	14-73
Syntax Rule	14-73
General Rules	14-73
TERMINATE Statement	14-74
Function	14-74
General Format	14-74
Syntax Rule	14-74
General Rules	14-74
USE BEFORE REPORTING Statement	14-76
Function	14-76
General Format	14-76
Syntax Rules	14-76
General Rules	14-76
Report Writer Sample Program	14-77

About This Chapter

This chapter describes the AIX VS COBOL Report Writer module facility. The Report Writer facility allows the COBOL programmer to produce a report by specifying the physical appearance of the report, rather than by specifying the detailed procedures necessary to produce that report. This chapter gives detailed information on Report Writer concepts, formats, syntax, and general rules.

Introduction

The report writer is a special purpose feature that places its emphasis on the organization, format, and contents of an output report. Although a report can be produced using the standard COBOL language, the report writer language feature provides a more concise facility for report structuring and report production. Much of the Procedure Division programming that would normally be supplied by the programmer is instead provided automatically by the report writer control system (RWCS). Thus, you are relieved of writing procedures for moving data, constructing print lines, counting lines on a page, numbering pages, producing heading and footing lines, recognizing the end of logical data subdivisions, updating sum counters, etc. All of these operations are accomplished by the RWCS as a consequence of source language statements that appear primarily in the REPORT SECTION of the Data Division of the source program.

REPORT SECTION

The REPORT SECTION of a COBOL Data Division contains one or more report description entries (RD entries), each of which forms the complete description of a report.

The report named in the report description entry is not assigned directly to an output file. Instead, it is associated with a file name in the FILE SECTION when an OPEN statement specifying the file name is executed. More than one report may be associated with the same file name and the CODE clause is used to differentiate among the reports. For an external file connector referenced by a file name, separately compiled programs may specify different reports for the same file name. The file description entry must specify the name of a report description entry for each report associated with a given file name in this program.

The report description entry contains a set of clauses that names the report and supplies specific information about the format of the printed page and the organization of the subdivisions of the report. An identification code may be given in the report description entry so that each report may be identified separately in an intermediate output file.

Following each report description entry are one or more 01 level-number entries, each followed by a hierarchical structure similar to COBOL record descriptions. Each 01 level-number entry and its subordinate entries describe a report group. Each report group consists of one or more print lines that are regarded as a unit. A report group is printed entirely on one logical page; it is never split across pages.

Report Structure

When structuring a report, major consideration must be given to vertical and horizontal spacing requirements, manipulation of data, and the physical and logical subdivisions of a report.

Vertical Spacing

The report writer feature allows you to describe report groups containing multiple lines. The vertical positioning of the lines on a page is specified by the LINE NUMBER clause that is associated with each line. The NEXT GROUP clause indicates how many lines to space after presenting the last line of the group. The first LINE NUMBER clause of the next group indicates additional spacing information to be used in positioning of that group.

Horizontal Spacing

The report writer allows the user to position the fields of data on a report line by means of the COLUMN NUMBER clause. The report writer control system supplies space fill between all defined fields.

Data Manipulation

When you use the report writer feature, data movement to a report group is directed by REPORT SECTION clauses rather than Procedure Division statements. The REPORT SECTION clauses that effect the manipulation of data are the SOURCE, SUM, and VALUE clauses.

The SOURCE clause specifies the sending data-item of an implicit MOVE statement. The receiving printable item is defined by the description of the report group item in which the SOURCE clause appears.

The SUM clause automatically causes the establishment of a sum counter. The object of the SUM clause names the data item(s) that are added to the sum counter when a GENERATE statement is executed. The movement of the sum counter contents to the receiving printable item, defined by the description of the report group item in which the SUM clause appears, is accomplished automatically when that report group is presented.

The VALUE clause defines a literal that appears in the printable item of a report group each time that report group is to be presented.

In summary, a data-item in a report group is presented only if it has a COLUMN NUMBER clause specifying where it is to be presented. The value that is placed in a printable item is determined by the SOURCE, SUM, or VALUE clause stated in the report group description. Under no circumstances can a report group printable item receive a value directly through a Procedure Division statement.

Report Subdivisions

The physical and logical organization of a report interact to determine which information is presented on a page.

Physical Subdivision of a Report

The PAGE clause specifies the length of the page, the size of the heading and footing areas, and the size of the area in which the detail lines will appear. The report writer control system uses the LINE NUMBER and NEXT GROUP clauses to position these report groups, and when necessary, to advance to a new page with automatic production of PAGE HEADING and PAGE FOOTING report groups.

Logical Subdivisions of a Report

DETAIL report groups may be structured into a nested set of control groups. Each control group can begin with a CONTROL HEADING report group and end with a CONTROL FOOTING report group.

When nested control groups are defined, the recognition of a change in value of a control data-item in a control hierarchy is called a control break. The heading and footing lines associated with the control data name are called CONTROL HEADING and CONTROL FOOTING report groups.

During the execution of a `GENERATE` statement, the report writer control system uses the control hierarchy to check automatically for control breaks. If a control break has occurred, all controls that are minor to it are considered to have changed, even though they may not in fact have changed. The occurrence of a control break causes the following sequence of events to take place:

1. ALL CONTROL FOOTING report groups are presented up to, and including, the one at the level at which the control break occurred.
2. ALL CONTROL HEADING report groups are presented from the control break level down to the most minor control.
3. The `DETAIL` report group named in the `GENERATE` statement is presented.

Procedure Division Report Writer Statements

The report writer statements that appear in the Procedure Division are: `INITIATE`, `GENERATE`, `TERMINATE`, `SUPPRESS`, and `USE BEFORE REPORTING`.

The `INITIATE` statement causes the report writer control system to perform automatically a number of initialization functions. A report must be initiated before any detail processing may take place.

The `GENERATE` statement which specifies a data-name causes the named `DETAIL` report group to be formatted and written to the output device. In addition, it causes the report writer control system to perform the many implicit actions described in the preceding section.

The `GENERATE` statement which specifies a report-name provides a means of summary reporting. A report produced by this type of statement has all detail print lines suppressed automatically and consists of only the summary totals accumulated during the processing of the `DETAIL` report group. The report writer control system processing for a `GENERATE` report-name statement is identical to that which occurs for a `GENERATE` data-name statement, except that the former results in the suppression of detail print lines.

The `TERMINATE` statement causes the report writer control system to perform all of the automatic functions associated with the termination of a report. The `TERMINATE` statement must be executed before the file containing the report is closed.

The `SUPPRESS` statement provides the object time facility to suppress the printing of an entire report group.

The `BEFORE REPORTING` phrase of the `USE` statement provides a mechanism whereby the Procedure Division statement may be executed at specific instances within the automatic procedures performed by the report writer control system. The statements in the `USE BEFORE REPORTING` procedure may alter the contents of the data items that are referenced by `SOURCE` clauses. Thus control is possible over the contents of data items referenced within report groups that are produced automatically.

Language Concepts

This section describes COBOL language concepts.

Report File

A report file is an output file having sequential organization. A report file has a file description entry containing a `REPORT` clause. The contents of a report file consists of records that are written under control of the report writer control system (RWCS).

A report file is named by a file control entry and is described by a file description entry containing a `REPORT` clause. A report file is referred to and accessed by the `OPEN`, `GENERATE`, `INITIATE`, `SUPPRESS`, `TERMINATE`, `USE BEFORE REPORTING`, and `CLOSE` statements.

Special Register PAGE-COUNTER

The reserved word PAGE-COUNTER is a name for a page counter that is generated for each report description entry in the REPORT SECTION of the Data Division. The implicit description is that of an unsigned integer that must be capable of representing a range of values from 1 through 999999. The usage is defined by the implementer. The value in PAGE-COUNTER is maintained by the report writer control system (RWCS) and is used by the program to number the pages of a report. PAGE-COUNTER may be referenced only in the SOURCE clause of the report section and in Procedure Division statements. Refer to "PAGE-COUNTER Rules" on page 14-18.

Special Register LINE-COUNTER

The reserved word LINE-COUNTER is a name for a line counter that is generated for each report description entry in the REPORT SECTION of the Data Division. The implicit description is that of an unsigned integer that must be capable of representing a range of values from 0 through 999999. The usage is defined by the implementer. The value in LINE-COUNTER is maintained by the RWCS, and is used to determine the vertical positioning of a report. LINE-COUNTER may be referenced only in the SOURCE clause of the report section and in Procedure Division statements. However, only the RWCS may change the value of LINE-COUNTER. Refer to "LINE-COUNTER Rules" on page 14-18.

Special Register PRINT-SWITCH

The reserved word PRINT-SWITCH is a name for a register whose value may be set nonzero in the course of a USE BEFORE REPORTING declarative procedure. This has the effect of suppressing printing of the corresponding report group.

OSVS

Subscripting

In the REPORT SECTION neither a sum counter nor the special registers LINE-COUNTER and PAGE-COUNTER can be used as a subscript.

Environment Division in the Report Writer Module

INPUT-OUTPUT Section

Refer to Chapter 8, "File Input and Output" for information concerning the INPUT-OUTPUT SECTION.

FILE-CONTROL Paragraph

Function

The FILE-CONTROL entry declares the relevant physical attributes of a report file. Refer to Chapter 8, "File Input and Output" on page 8-1 for information concerning the FILE-CONTROL paragraph.

Additional Syntax Rule

A report file must have sequential organization. Each report file specified in the SELECT clause must have a file description entry containing a REPORT clause in the Data Division of the same program.

I-O-CONTROL Paragraph

Function

Refer to Chapter 8, "File Input and Output" for information on the I-O-CONTROL paragraph.

Additional Syntax Rule

A report file may not appear in a SAME clause for which the RECORD phrase is specified.

Data Division in the Report Writer Module

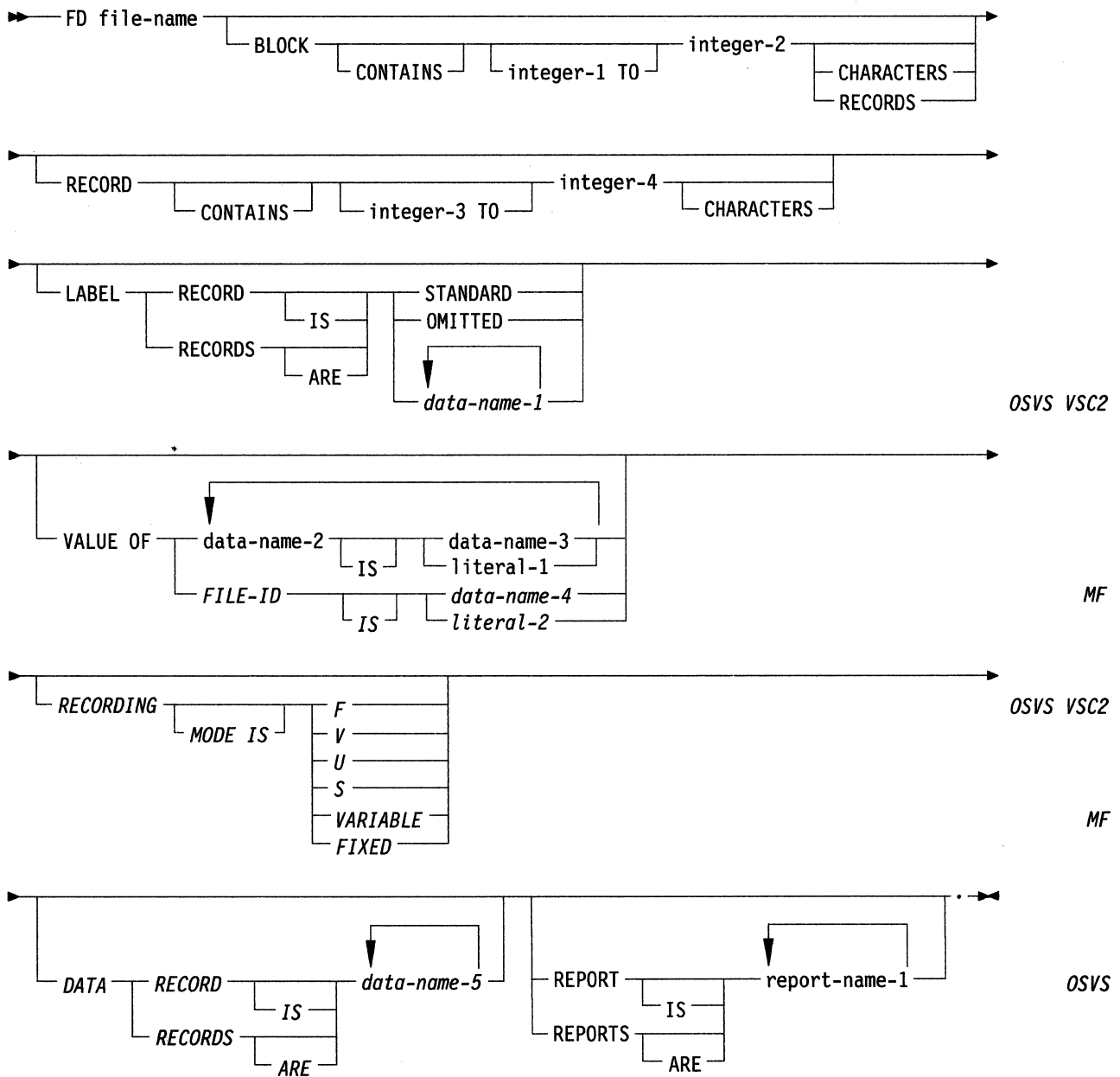
File Description Entry

Function

The file description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

General Format

The following figure shows the format of the file description entry:



Syntax Rules

The following syntax rules apply to the file description entry:

1. The level indicator FD identifies the beginning of the file description entry for a report file and must precede the file name of the report file.
2. The clauses that follow file-name may appear in any order.
3. file-name may only reference a sequential file.
4. No record description entries may follow the file description entry for a report file. *Record description entries are permitted.* OSVS
5. The subject of a file description entry that specifies a REPORT clause may be referenced in the Procedure Division only by the USE statement, the CLOSE statement, or the OPEN statement with the OUTPUT or EXTEND phrase.

General Rules

The following rules apply to the file description entry:

1. With the exception of the REPORT clause, all clauses within the file description entry for a report file are presented within the sequential I-O module.
2. Refer to the following section "REPORT Clause" on page 14-14 for details on the REPORT clause.

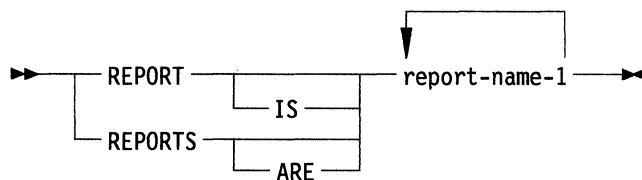
REPORT Clause

Function

The REPORT clause specifies the names of reports that comprise a report file.

General Format

The following figure shows the format of the REPORT clause:



Syntax Rules

The following syntax rules apply to the REPORT clause:

1. Each report-name specified in a REPORT clause must be the subject of a report description entry in the REPORT SECTION of the same program. The order of appearance of the report-names is not significant.
2. A report-name must appear in only one REPORT clause.
3. The subject of a file description entry that specifies a REPORT clause may be referenced in the Procedure Division only by the USE statement, the CLOSE statement, or the OPEN statement with the OUTPUT or EXTEND phrase.

General Rules

The following rules apply to the REPORT clause:

1. The presence of more than one report-name in a REPORT clause indicates that the file contains more than one report.
2. After execution of an INITIATE statement and before the execution of a TERMINATE statement for the same report file, the report file is under the control of the RWCS. While a report file is under the control of the RWCS, no input-output statement may be executed which references that report file.

Example

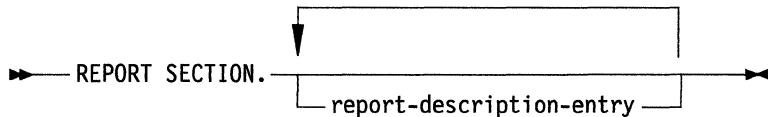
The following is an example of the REPORT clause:

```
FILE SECTION.  
:  
:  
FD  REPORT-FILE  
    LABEL RECORDS OMITTED  
    RECORD CONTAINS 121 CHARACTERS  
    REPORT IS EXPENSE-REPORT.
```

REPORT SECTION

The REPORT SECTION is located in the Data Division of a source program. This section describes the reports written onto report files. The description of each report must begin with a report description entry (RD entry) and be followed by one or more report group description entries.

The general format of the REPORT SECTION is:



Report Description Entry

In addition to naming the report, the RD entry defines the format of each page of the report by specifying the vertical boundaries of the region within which each type of report group may be printed. The report description entry also specifies the control data items. When the report is produced, changes in the values of the control data items cause the detail information of the report to be processed in groups called control groups.

Each report named in the REPORT clause of a file description entry in the FILE SECTION must be the subject of a report description entry in the REPORT SECTION. Furthermore, each report in the REPORT SECTION must be named in one and only one file description entry.

Report Group Description Entry

The report groups that will comprise the report are described following the report description entry. The description of each report group begins with a report group description entry (an entry that has the 01 level-number and a TYPE clause). Subordinate to the report group description entry, group and elementary entries may be used to further describe the characteristics of the report group.

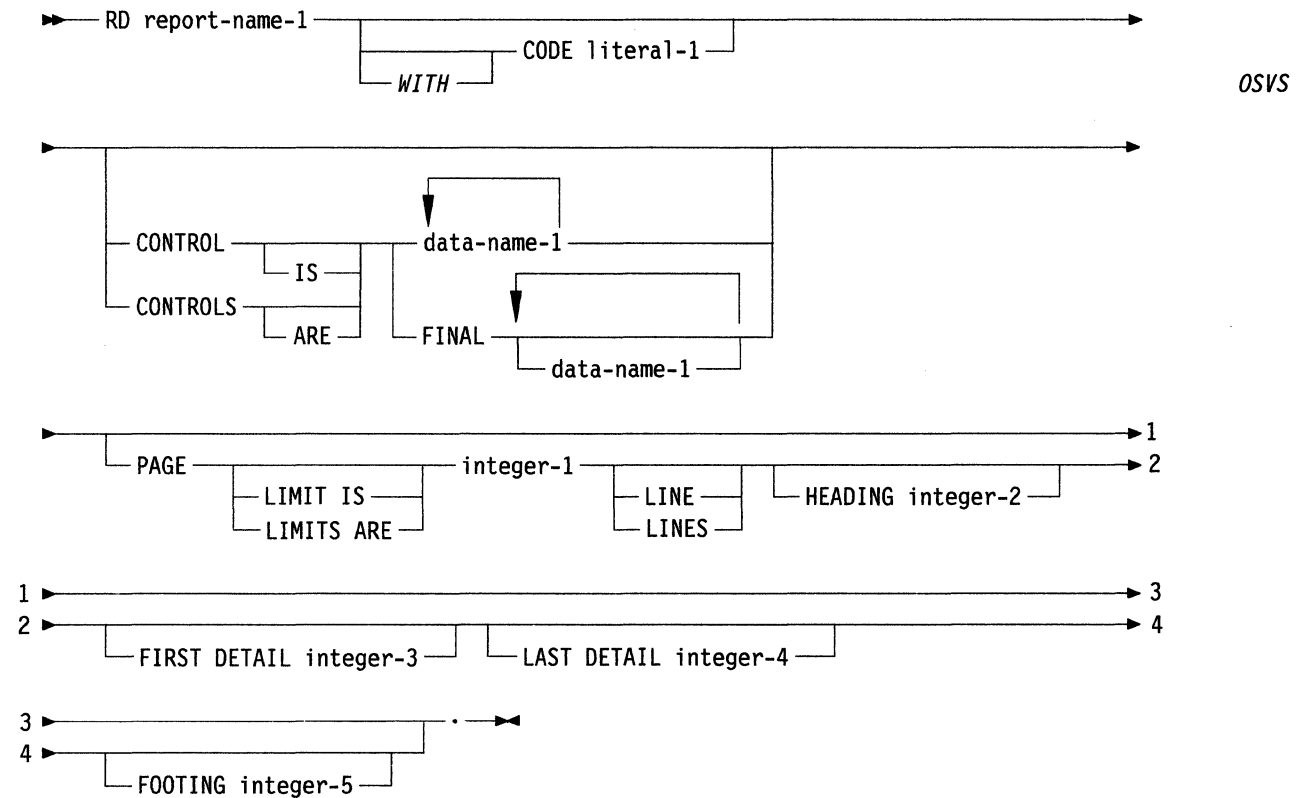
REPORT Description Entry

Function

The report description entry names a report, specifies any identifying characters to be appended to each print line in the report and describes the physical structure and organization of that report.

General Format

The following figure shows the general format of the REPORT description entry:



Syntax Rules

The following syntax rules apply to the REPORT entry:

1. report-name-1 must appear in one and only one REPORT clause.
2. The order of appearance of the clauses following report-name-1 is immaterial.
3. report-name-1 is the highest permissible qualifier that may be specified for LINE-COUNTER, PAGE-COUNTER, and all data-names defined within the REPORT SECTION.

General Rule

The following paragraphs list rules for CODE clause, CONTROL clause, and PAGE clause in alphabetical order.

PAGE-COUNTER Rules

The following rules apply to PAGE-COUNTER:

1. PAGE-COUNTER is the reserved word used to reference a special register that is automatically created for each report specified in the REPORT SECTION. Refer to "Special Register PAGE-COUNTER" on page 14-9.
2. In the REPORT SECTION, a reference to PAGE-COUNTER can only appear in a SOURCE clause. In the Procedure Division, PAGE-COUNTER may be used in any context in which a data-item with an integer value can appear.
3. If more than one PAGE-COUNTER exists in a program, PAGE-COUNTER must be qualified by a report-name whenever it is referenced in the Procedure Division.

In the REPORT SECTION an unqualified reference to PAGE-COUNTER is qualified implicitly by the name of the report in whose report description entry the reference is made. When the PAGE-COUNTER of a different report is referenced, PAGE-COUNTER must be explicitly qualified by the report-name associated with the different report.

4. Execution of the INITIATE statement causes the report writer control system to set the PAGE-COUNTER of the referenced report to one.
5. PAGE-COUNTER is automatically incremented by one each time the report writer control system executes a page advance.
6. PAGE-COUNTER may be altered by Procedure Division statements.

LINE-COUNTER Rules

The following rules apply to LINE-COUNTER:

1. LINE-COUNTER is the reserved word used to reference a special register that is automatically created for each report specified in the REPORT SECTION. Refer to "Special Register LINE-COUNTER" on page 14-9.
2. In the REPORT SECTION a reference to LINE-COUNTER can only appear in a SOURCE clause. In the Procedure Division, LINE-COUNTER may be used in any context in which a data-item with an integral value may appear. However, only the RWCS can change the content of LINE-COUNTER.
3. If more than one LINE-COUNTER exists in a program, LINE-COUNTER must be qualified by a report-name whenever it is referenced in the Procedure Division.

In the REPORT SECTION an unqualified reference to LINE-COUNTER is qualified implicitly by the name of the report in whose description entry the reference is made. When the LINE-COUNTER of a different report is referenced, LINE-COUNTER must be explicitly qualified by the report-name associated with the different report.

4. Execution of an INITIATE statement causes the report writer control system to set the LINE-COUNTER of the referenced report to zero. The RWCS also automatically resets LINE-COUNTER to zero each time it executes a page advance.
5. The value of LINE-COUNTER is not affected by the processing of nonprintable report groups or by the processing of a printable report group whose printing is suppressed by means of the SUPPRESS statement.
6. At the time each print line is presented, the value of LINE-COUNTER represents the line number on which the print line is presented. The value of LINE-COUNTER after the presentation of a report group is governed by the presentation rules for the report group. Refer to "Presentation Rules Tables" on page 14-32.

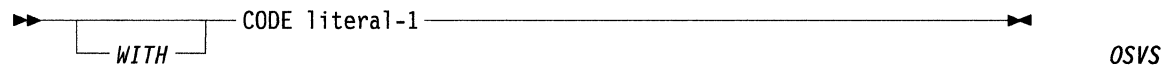
CODE Clause

Function

The CODE clause specifies a two-character literal that identifies each print line as belonging to a specific report.

General Format

The following figure shows the general format of the CODE clause:



Syntax Rules

The following syntax rules apply to the CODE clause:

1. literal-1 must be a two-character nonnumeric literal.
2. If the CODE clause is specified for any report in a file, it must be specified for all reports in that file.

General Rules

The following general rules apply to the CODE clause:

1. When the CODE clause is specified, literal-1 is automatically placed in the first two character positions of each report writer logical record.
2. The positions occupied by literal-1 are not included in the description of the print line, but are included in the logical record size.

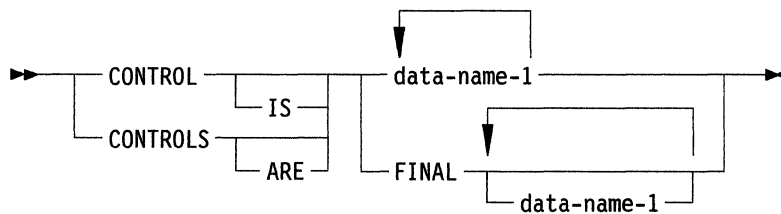
CONTROL Clause

Function

The CONTROL clause establishes the levels of the control hierarchy for the report.

General Format

The following figure shows the general format of the CONTROL clause:



Syntax Rules

The following syntax rules apply to the CONTROL clause:

1. data-name-1 must not be defined in the REPORT SECTION. data-name-1 may be qualified.
2. Each recurrence of data-name-1 must identify a different data item.
3. data-name-1 must not have subordinate to it a variable occurrence data item.

General Rules

The following general rules apply to the CONTROL clause:

1. data-name-1 and the word FINAL specify the levels of the control hierarchy. FINAL, if specified, is the highest control. data-name-1 is the major control. The next recurrence of data-name-1 is an intermediate control, and so on. The last recurrence of data-name-1 is the minor control.
2. The execution of the chronologically first GENERATE statement for a given report causes the RWCS to save the values of all control data items associated with that report. On subsequent executions of all GENERATE statements for that report, control data items are tested by the RWCS for a change of value. A change of value in any control data-item causes a control break to occur. This control break is associated with the highest level for which a change of value is noted. Refer to "GENERATE Statement" on page 14-68.

-
3. The RWCS tests for a control break by comparing the content of each control data-item with the prior content of each control data-item that was saved when the previous GENERATE statement for the same report was executed. The RWCS applies the inequality relation test as follows:
 - a. If the control data-item is a numeric data item, the relation test is for the comparison of two numeric operands.
 - b. If the control data-item is an index data item, the relation test is for the comparison of two index data items.
 - c. If the control data-item is a data item other than as described in 3a and 3b, the relation test is for the comparison of two nonnumeric operands.

The inequality relation test is further explained in the appropriate paragraph. Refer to "Relation Condition" on page 7-9.

4. FINAL is used when the most inclusive control group in the report is not associated with a control data-name.

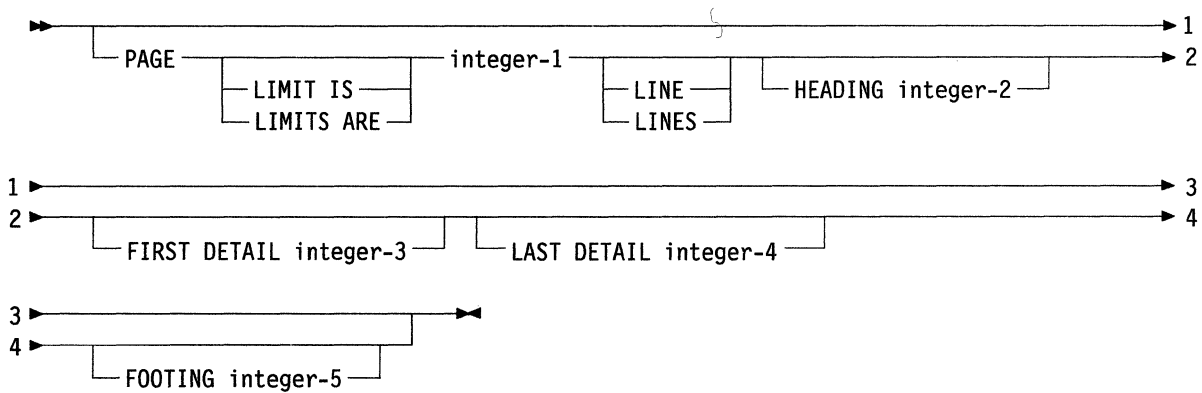
PAGE Clause

Function

The PAGE clause defines the length of a page and the vertical subdivisions within which report groups are presented.

General Format

The following figure shows the format of the PAGE clause:



Syntax Rules

The following syntax rules apply to the PAGE clause:

1. The HEADING, FIRST DETAIL, LAST DETAIL, and FOOTING phrases may be written in any order.
2. integer-1 must not exceed three significant digits in length.
3. integer-2 must be greater than or equal to one.
4. integer-3 must be greater than or equal to integer-2.
5. integer-4 must be greater than or equal to integer-3.
6. integer-5 must be greater than or equal to integer-4.
7. integer-1 must be greater than or equal to integer-5.

-
8. The following rules indicate the vertical subdivision of the page in which each type of report group may appear when the PAGE clause is specified. Refer to "Page Regions" on page 14-25.
 - a. A report heading report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

A report heading report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.
 - b. A page heading report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.
 - c. A control heading or detail report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-4, inclusive.
 - d. A control footing report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-5, inclusive.
 - e. A page footing report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1, inclusive.
 - f. A report footing report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

A report footing report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1, inclusive.
 9. All report groups must be described so that they can be presented on one page. The RWCS never splits a multiline report group across page boundaries.

General Rules

The following general rules apply to the PAGE clause:

1. The vertical format of a report page is established using the integer values specified in the PAGE clause.
 - a. PAGE LIMIT integer-1 defines the size of a report page by specifying the number of lines available on each page.
 - b. HEADING integer-2 defines the first line number on which a report heading or page heading report group may be presented.
 - c. FIRST DETAIL integer-3 defines the first line number on which a body group may be presented. Report heading and page heading report groups may not be presented on or beyond the line number specified by integer-3.
 - d. LAST DETAIL integer-4 defines the last line number on which a control heading or detail report group may be presented.
 - e. FOOTING integer-5 defines the last line number on which a control footing report group may be presented. Page footing and report footing report groups must follow the line number specified by integer-5.

Figure 14-1 is an illustration of PAGE clause ranges:

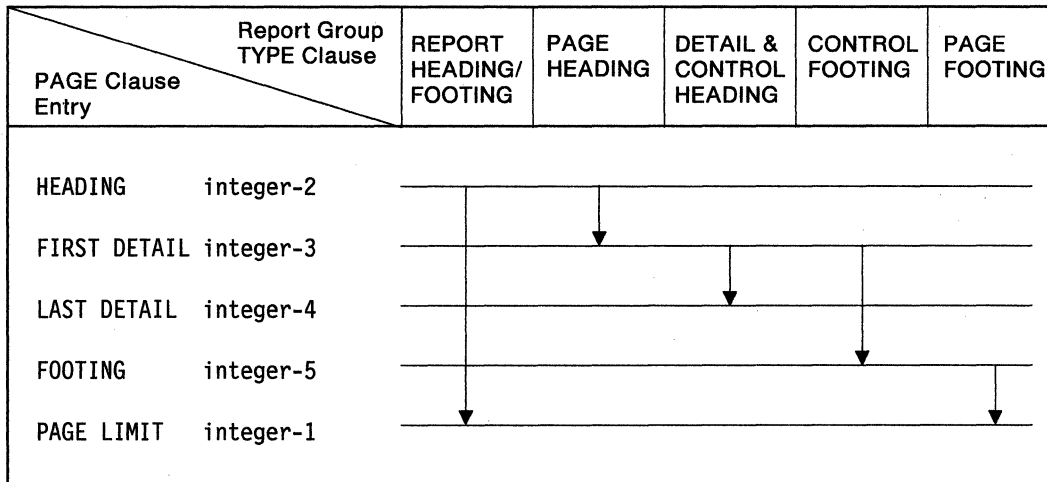


Figure 14-1. Illustration of PAGE Clause Ranges

2. If the PAGE clause is specified, the following implicit values are assumed for any omitted phrases:
 - a. If the HEADING phrase is omitted, a value of one is assumed for integer-2.
 - b. If the FIRST DETAIL phrase is omitted, a value equal to integer-2 is given to integer-3.
 - c. If the LAST DETAIL and the FOOTING phrases are both omitted, the value of integer-1 is given to both integer-4 and integer-5.
 - d. If the FOOTING phrase is specified and the LAST DETAIL phrase is omitted, the value of integer-5 is given to integer-4.
 - e. If the LAST DETAIL phrase is specified and the FOOTING phrase is omitted, the value of integer-4 is given to integer-5.

Figure 14-2 shows the default value for omitted PAGE LIMIT entries:

Omitted PAGE LIMIT Entry	Default Value Assumed
HEADING integer-2	integer-2 = 1
FIRST DETAIL integer-3	integer-3 = integer-2
LAST DETAIL integer-4	integer-4 = integer-5
FOOTING integer-5	integer-5 = integer-4
both LAST DETAIL integer-4 and FOOTING integer-5	integer-4 & integer-5 = integer-1

Figure 14-2. Values Assumed for Omitted PAGE Clause Options

3. If the PAGE clause is omitted, the report consists of a single page of indefinite length.
4. The presentation rules for each type of report group are specified in the appropriate paragraph. Refer to "Presentation Rules Tables" on page 14-32.

Page Regions

Page regions that are established by the PAGE clause are described in Table 14-1.

Report Groups That May Be Presented in the Region	First Line Number of the Region	Last Line Number of the Region
Report heading described with NEXT GROUP NEXT PAGE Report footing described with LINE integer-1 NEXT PAGE	integer-2	integer-1
Report heading not described with NEXT GROUP NEXT PAGE Page heading	integer-2	integer-3 minus 1
Control heading Detail	integer-3	integer-4
Control footing	integer-3	integer-5
Page footing Report footing not described with LINE integer-1 NEXT PAGE	integer-5	integer-1 plus 1

Example 1

The following example shows a sample Report Section:

```
DATA DIVISION.  
FILE SECTION.  
.  
.  
FD REPORT-FILE  
  REPORT IS EXPENSE-REPORT.  
.  
.  
REPORT SECTION.  
RD EXPENSE-REPORT  
  CONTROLS ARE FINAL QUARTER MM DD  
  PAGE LIMIT IS 59 LINES  
  HEADING      1  
  FIRST DETAIL  9  
  LAST DETAIL   48  
  FOOTING      52.
```


Example 2

The following example shows the RD entry PAGE clause and the resulting report lines.

ACME MANUFACTURING COMPANY							
QUARTERLY EXPENDITURES REPORT							
JANUARY EXPENDITURES							
MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST	
JANUARY	01	A00	2	A	2.00		
		A02	1	A	1.00		
		A02	2	C	16.00		
PURCHASES AND COST FOR 1-01			5		\$19.00	\$19.00	

JANUARY	02	A01	2	B	2.00		
		A04	10	A	10.00		
		A04	10	C	80.00		
PURCHASES AND COST FOR 1-02			22		\$92.00	\$111.00	

JANUARY	05	A01	2	B	2.00		
PURCHASES AND COST FOR 1-05			2		\$2.00	\$113.00	

JANUARY	08	A01	10	A	10.00		
		A01	8	B	12.48		
		A01	20	D	38.40		
PURCHASES AND COST FOR 1-08			38		\$60.88	\$173.88	

JANUARY	13	A00	4	B	6.24		
		A00	1	C	8.00		
PURCHASES AND COST FOR 1-13			5		\$14.24	\$188.12	

JANUARY	15	A00	10	D	19.20		
		A02	1	C	8.00		
PURCHASES AND COST FOR 1-15			11		\$27.20	\$215.32	

JANUARY	21	A03	10	E	30.00		
		A03	10	F	25.00		
		A03	10	G	50.00		
PURCHASES AND COST FOR 1-21			30		\$105.00	\$320.32	

JANUARY	23	A00	5	A	5.00		
PURCHASES AND COST FOR 1-23			5		\$5.00	\$325.32	

RD EXPENSE-REPORT
 CONTROLS ARE FINAL MONTH DAY
 PAGE LIMIT IS 59 LINES
 HEADING 1
 FIRST DETAIL 9
 LAST DETAIL 48
 FOOTING 52.

PAGE clause specifies:

1. Physical page depth
2. Heading area
3. Area in which detail lines may appear
4. Area in which footing lines may appear

Example 3

The following example shows the RD entry CONTROL clause and the resulting report lines.

JANUARY EXPENDITURES (CONTINUED)									
RD	EXPENSE-REPORT	MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST	
	CONTROLS ARE FINAL	JANUARY	26	A04	5	A	5.00		
	MONTH DAY-1			A04	5	B	7.80		
	PAGE LIMIT IS	PURCHASES AND COST FOR 1-26 10						\$12.80	\$338.12
	59 LINES	*****							
	HEADING 1	JANUARY	27	A00	6	B	9.36		
	FIRST DETAIL 9			A00	15	C	120.00		
	LAST DETAIL 48	PURCHASES AND COST FOR 1-27 21						\$129.36	\$467.48
	FOOTING 52.	*****							
		JANUARY	30	A00	2	B	3.12		
				A02	10	A	10.00		
				A02	1	C	8.00		
				A04	15	B	23.40		
				A04	10	C	80.00		
		PURCHASES AND COST FOR 1-30 38						\$124.52	\$592.00

		JANUARY	31	A00	1	A	1.00		
				A04	6	A	6.00		
		PURCHASES AND COST FOR 1-31 7						\$7.00	\$599.00

		TOTAL COST FOR JANUARY WAS						\$599.00	

CONTROL clause specifies that control breaks occur when:

1. DAY report field changes value
2. MONTH report field changes value
3. FINAL report field changes value that is when end-of-report is reached. (Not shown on this page.)

1

2

Report Group Description Entry

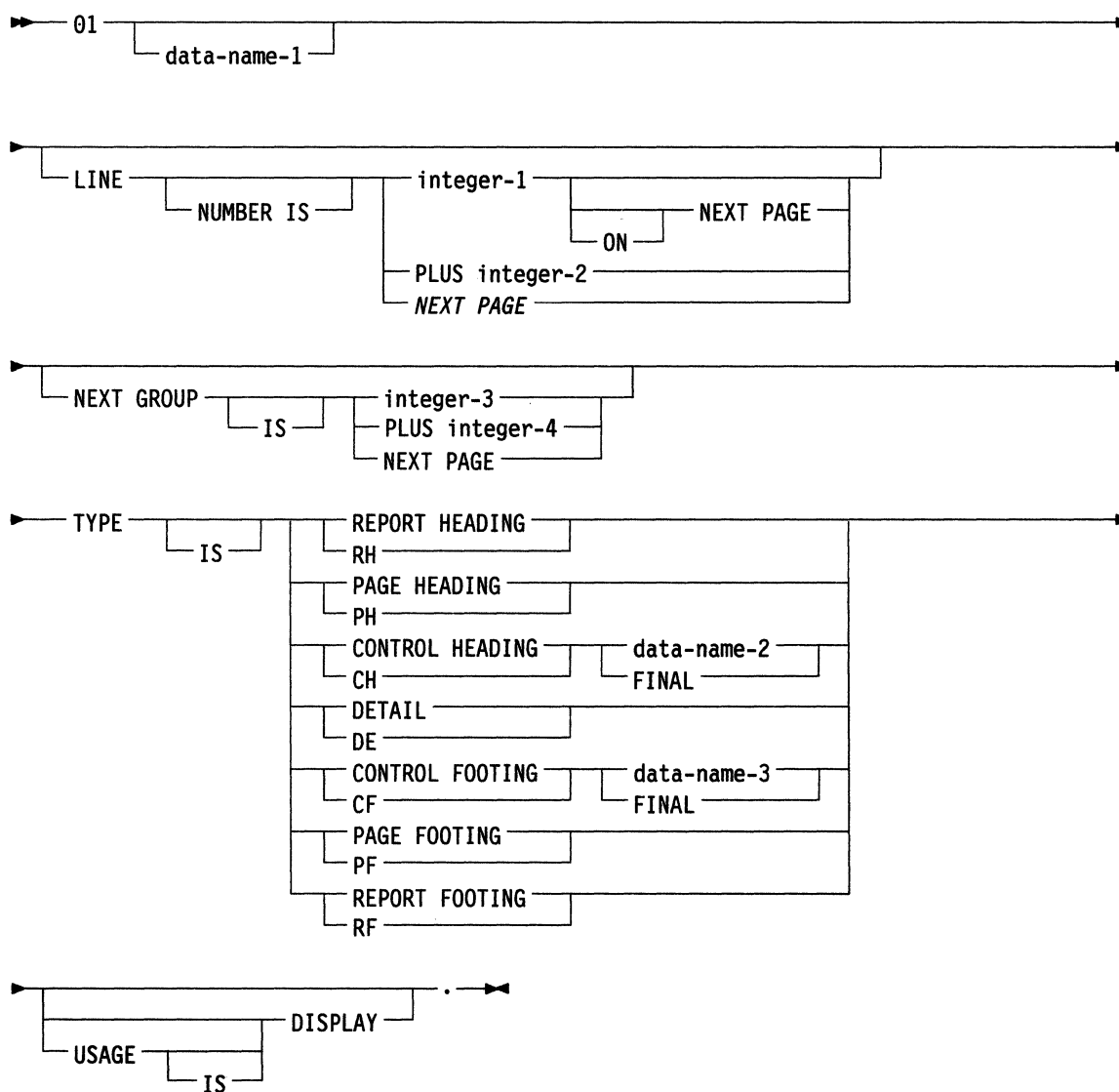
Function

The report group description entry specifies the characteristics of a report group and of the individual items within a report group.

General Format

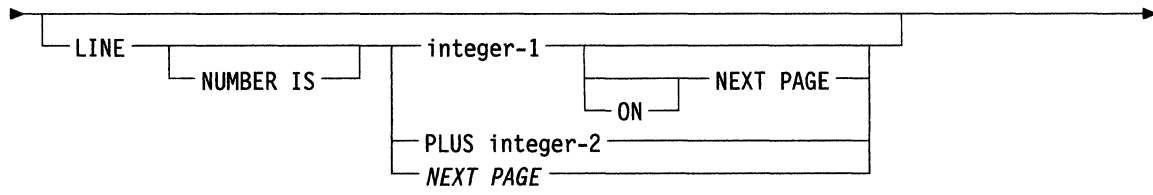
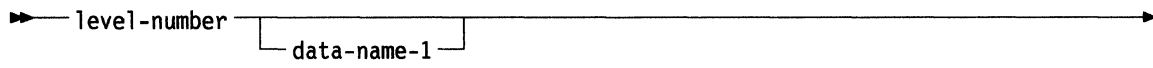
The following figures show the format for the report group description entry:

Format 1

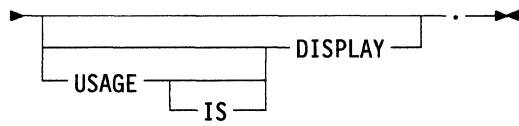


OSVS

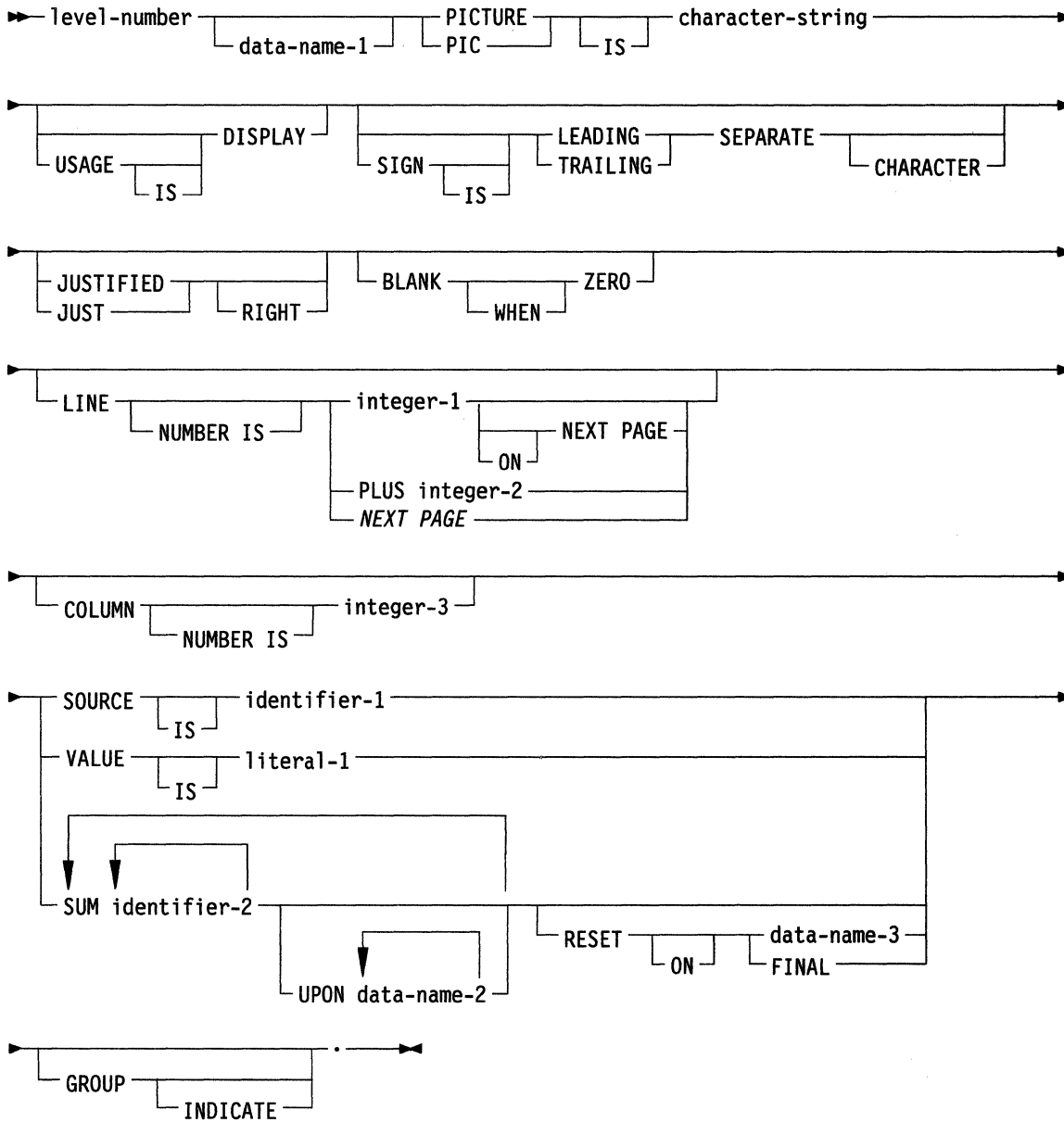
Format 2



OSVS



Format 3



OSVS

Syntax Rules

The following syntax rules apply to the report description entry:

1. The report group description entry can appear only in the **REPORT SECTION**.
2. Except for the data-name clause, which when present must immediately follow the level-number, the clauses may be written in any sequence.
3. In Format 2 the level-number may be any integer from 02 to 48 inclusive. In Format 3 the level-number may be any integer from 02 to 49 inclusive.
4. A description of a report group may consist of one, two, or three hierarchical levels:
 - a. The first entry that describes a report group must be a Format 1 entry.
 - b. Both Format 2 and Format 3 entries may be immediately subordinate to a Format 1 entry.
 - c. At least one Format 3 entry must be immediately subordinate to a Format 2 entry.
 - d. Format 3 entries must define elementary data items.
5. In a Format 1 entry, data-name-1 is required only when:
 - a. A detail report group is referenced by a **GENERATE** statement.
 - b. A detail report group is referenced by the **UPON** phrase of a **SUM** clause.
 - c. A report group is referenced in a **USE BEFORE REPORTING** sentence.
 - d. The name of a control footing report group is used to qualify a reference to a sum counter.

If specified, data-name-1 may be referenced only by a **GENERATE** statement, the **UPON** phrase of a **SUM** clause, a **USE BEFORE REPORTING** sentence, or as a sum counter qualifier.

6. A Format 2 entry must contain at least one optional clause.
7. In a Format 2 entry, data-name-1 is optional. If present, it may be used only to qualify a sum counter reference.
8. In the **REPORT SECTION**, the **USAGE** clause is used only to declare the usage of printable items.
 - a. If the **USAGE** clause appears in a Format 3 entry, that entry must define a printable item.
 - b. If the **USAGE** clause appears in a Format 1 or Format 2 entry, at least one subordinate entry must define a printable item.
9. An entry that contains a **LINE NUMBER** clause must not have a subordinate entry that also contains a **LINE NUMBER** clause. *However, an entry containing the **LINE NUMBER NEXT PAGE** clause may have a subordinate entry containing a **LINE NUMBER** clause but without the **NEXT PAGE** option.*

OSVS

10. In Format 3:

- a. A GROUP INDICATE clause may appear only in a type detail report group.
- b. A SUM clause may appear only in a type control footing report group.
- c. An entry that contains a COLUMN NUMBER clause but no LINE NUMBER clause must be subordinate to an entry that contains a LINE NUMBER clause.
- d. data-name-1 is optional but may be specified in any entry. data-name-1 may be referenced only if the entry defines a sum counter.
- e. LINE NUMBER clause must not be the only clause specified.
- f. An entry that contains a VALUE clause must also have a COLUMN NUMBER clause.

11. Table 14-2 shows all permissible clause combinations for a Format 3 entry. Read the table from left to right along the selected row.

- An M indicates that the presence of the clause is mandatory.
- A P indicates that the presence of the clause is permitted, but not required.
- A blank indicates that the clause is not permitted.

Pic	Column	Source	Sum	Value	Just	Blank When Zero	Group Indicate	Usage	Sign	Line
M			M						P	P
M	M		M			P		P	P	P
M	P	M			P		P	P	P	P
M	P	M				P	P	P	P	P
M	M			M	P		P	P	P	P

General Rules

The following general rules apply to the report group entry:

1. Format 1 is the report group entry. The report is defined by the contents of this entry and all of its subordinate entries.
2. The BLANK WHEN ZERO clause, the JUSTIFIED clause, and the PICTURE clause for the Report Writer module are the same as the BLANK WHEN ZERO clause, the JUSTIFIED clause, and the PICTURE clause in the Nucleus. Refer to Chapter 3, "Introduction to the Nucleus" for specifications. The other clauses of the report group description entry are presented in alphabetical order later in this chapter.

Presentation Rules Tables

The tables and rules on the following pages specify:

1. The permissible combinations of LINE NUMBER and NEXT GROUP clauses for each type of report group.
2. The requirements that are placed on the use of these clauses.
3. The interpretation that the RWCS gives to these clauses.

Organization

Presentation rules tables follow for each of these report groups: report heading, page heading, page footing, and report footing. In addition, detail report groups, control heading report groups, or control footing report groups are treated jointly in the body group presentation rule table. Refer to “Body Group Presentation Rules” on page 14-38.

Columns 1 and 2 of a presentation rules table list all of the permissible combinations of LINE NUMBER and NEXT GROUP clauses for the designated report group type. Consequently, to identify the set of presentation rules that apply to a particular combination of LINE NUMBER and NEXT GROUP clauses, a presentation rules table is read from left to right, along the selected row.

The applicable rules columns of a presentation rules table contain two parts. The first part specifies the rules that apply if the report description contains a PAGE clause. The second part specifies the rules that apply if the PAGE clause is omitted. The purpose of the rules in the rules columns is discussed below:

1. Upper limit rules and lower limit rules

These rules specify the vertical subdivisions of the page within which the specified report group may be presented.

In the absence of a PAGE clause, the printed report is not considered to be partitioned into vertical subdivisions. Consequently, within the tables no upper limit rule or lower limit rule is specified for a report description in which the PAGE clause is omitted.

2. Fit test rules

The fit test rules are applicable only to body groups and are specified only within the body group presentation rules table. At object time the RWCS applies the fit test rules to determine whether the designated body group can be presented on the page to which the report is currently positioned.

However, even for body groups, there are no fit test rules when the PAGE clause is omitted from the report description entry.

3. First print line position rules

The first print line position rules specify where on the report medium the RWCS shall present the first print line of the given report group.

The presentation rules tables do not specify where on the report medium the RWCS shall present the second and subsequent print lines (if any) of a report group. Certain general rules determine where the second and subsequent print lines of a report group shall be presented. Refer to “LINE NUMBER Clause” on page 14-49 for this information.

4. Next group rules

The next group rules relate to the proper use of the NEXT GROUP clause.

5. Final LINE-COUNTER setting rules:

The terminal values that the RWCS places in LINE-COUNTER after presenting report groups are specified by the final LINE-COUNTER setting rules.

LINE NUMBER Clause Notation

Column 1 of the presentation rules table uses a shorthand notation to describe the sequence of LINE NUMBER clauses that may appear in the description of a report group. The meaning of the abbreviations used in column 1 is as follows:

1. The letter A represents one or more absolute LINE NUMBER clauses, none of which has the NEXT PAGE phrase, that appear in consecutive order within the sequence of LINE NUMBER clauses in the report group description entry.
2. The letter R represents one or more relative LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the report group description entry.
3. The letters NP represent one or more absolute LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the report group description entry with the phrase NEXT PAGE appearing in the first and only in the first LINE NUMBER clause. *This entry may appear without a line number.* OSVS

When two abbreviations appear together, they refer to a sequence of LINE NUMBER clauses that consist of the two specified consecutive sequences. For example, A R refers to a report group description entry within which the A sequence (defined in 1) is immediately followed by the R sequence (defined in 2).

LINE NUMBER Clause Sequence Substitutions

Where A R is shown to be a permissible sequence in the presentation rules table, A is also permissible and the same presentation rules are applicable.

Where NP R is shown to be a permissible sequence in the presentation rules table, NP is also permissible and the same presentation rules are applicable.

Saved Next Group Integer Description

Saved next group integer is a data-item that is addressable only by the RWCS. When an absolute NEXT GROUP clause specifies a vertical positioning value that cannot be accommodated on the current page, the RWCS stores that value in saved next group integer. After page advance processing, the RWCS positions the next body group using the value stored in saved next group integer.

Report Heading Group Presentation Rules

Figure 14-3 on page 14-36 points to the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a report heading report group. The report heading group presentation rules are as follows:

1. Upper limit rule
The first line number on which the report heading report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.
2. Lower limit rules
 - a. The last line number on which the report heading report group can be presented is the line number that is obtained by subtracting 1 from the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.
 - b. The last line number on which the report heading report group can be presented is the line number specified by integer-1 of the PAGE clause.

-
3. First print line position rules
 - a. The first print line of the report heading report group is presented on the line number specified by the integer or its LINE NUMBER clause.
 - b. The first print line of the report heading report group is presented on the line number obtained by adding the integer of the first LINE NUMBER clause and the value obtained by subtracting 1 from the value of integer-2 of the HEADING phrase of the PAGE clause.
 - c. The report heading report group is not presented.
 - d. The first print line of the report heading report group is presented on the line number obtained by adding the content of its LINE-COUNTER (in this case, zero) to the integer of the first LINE NUMBER clause.
 4. Next group rules
 - a. The NEXT GROUP integer must be greater than the line number on which the final print line of the report heading report group is presented. In addition, the NEXT GROUP integer must be less than the line number specified by the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.
 - b. The sum of the NEXT GROUP integer and the line number on which the final print line of the report heading report group is presented must be less than the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.
 - c. NEXT GROUP PAGE signifies that the report heading report group is to be presented entirely by itself on the first page of the report. The RWCS processes no other report group while positioned to the first page of the report.
 5. Final LINE-COUNTER setting rules
 - a. After the report heading report group is presented, the RWCS places sum of the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.
 - b. After the report heading report group is presented, the RWCS places the sum of the NEXT GROUP integer and the line number on which the final print line of the report heading report group was presented into LINE-COUNTER as the final LINE-COUNTER setting.
 - c. After the report heading report group is presented, the RWCS places zero into LINE-COUNTER as the final LINE-COUNTER setting.
 - d. After the report heading report group is presented, the final LINE-COUNTER setting is the line number on which the final print line of the report heading report group was presented.
 - e. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

**		Applicable Rules ***						
		If the PAGE clause is specified.					If the PAGE clause is omitted.	
Sequence of LINE NUMBER clauses *	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting	First Print Line Position	Final LINE-COUNTER Setting
A R	Absolute	1	2a	3a	4a	5a	Illegal Combination †	
A R	Relative	1	2a	3a	4b	5b	Illegal Combination †	
A R	NEXT PAGE	1	2b	3a	4c	5c	Illegal Combination †	
A R		1	2a	3a		5d	Illegal Combination †	
R	Absolute	1	2a	3b	4a	5a	Illegal Combination ††	
R	Relative	1	2a	3b	4b	5b	3d	5d
R	NEXT PAGE	1	2b	3b	4c	5c	Illegal Combination ††	
R		1	2a	3b		5d	3c	5d
				3c		5e	3c	5e

Figure 14-3. Report Heading Group Presentation Rules Table

- * Refer to “LINE NUMBER Clause Notation” on page 14-34 for a description of the abbreviations used in column 1.
- ** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.
- *** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.
- † See the “LINE NUMBER Clause” on page 14-49.
- †† See the “NEXT GROUP Clause” on page 14-51.

Note: If the PAGE clause is omitted from the report description entry, then a page heading report group may not be defined. Refer to “TYPE Clause” on page 14-59.

Page Heading Group Presentation Rules

Figure 14-4 on page 14-37 points to the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a page heading report group.

**		Applicable Rules ***				
		If the PAGE clause is specified. ****				
Sequence of LINE NUMBER clauses*	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting
A R		1	2	3a		4a
R		1	2	3b		4b
				3c		4c

Figure 14-4. Page Heading Group Presentation Rules Table

- * Refer to “LINE NUMBER Clause Notation” on page 14-34 for a description of the abbreviations used in column 1.
- ** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.
- *** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.
- **** If the PAGE clause is omitted from the report description entry, then a page heading report group may not be defined. Refer to “TYPE Clause” on page 14-59.

The page heading group presentation rules are as follows:

1. Upper limit rule

If a report heading report group has been presented on the page on which the page heading report group is to be presented, then the first line number on which the page heading report group can be presented is one greater than the final LINE-COUNTER setting established by the REPORT HEADING.

Otherwise, the first line number on which the page heading report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

2. Lower limit rule

The last line number on which the page heading report group can be presented is the line number that is obtained by subtracting one from the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

3. First print line position rules

a. The first print line of the page heading report group is presented on the line number specified by the integer of its LINE NUMBER clause.

b. If a report heading report group has been presented on the page on which the page heading report group is to be present, then the sum of the final LINE-COUNTER setting established by the report heading report group and the integer of the first LINE NUMBER clause of the page heading report group defines the line number on which the first print line of the page heading report group is presented.

Otherwise the sum of the integer of the first LINE NUMBER clause of the page heading report group and the value obtained by subtracting one from the value of integer-2 of the HEADING phrase of the PAGE clause defines the line number on which the first print line of the page heading report group is presented.

c. The page heading report group is not presented.

-
4. Final LINE-COUNTER setting rules
 - a. The final LINE-COUNTER setting is the line number on which the final print line of the page heading report group was presented.
 - b. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

Body Group Presentation Rules

Figure 14-5 on page 14-39 specifies the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in control heading, detail, and control footing report groups. The body group presentation rules are as follows:

1. Upper limit rule

The first line number on which a body group can be presented is the line number specified by the FIRST DETAIL phrase of the PAGE clause.

2. Lower limit rules

The last line number on which a control heading report group or detail report group can be presented is the line number specified by the LAST DETAIL phrase of the PAGE clause.

The last line number on which a control footing report group can be presented is the line number specified by the FOOTING phrase of the PAGE clause.

3. Fit test rules

- a. If the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, then the body group is presented on the page to which the report is currently positioned.

Otherwise, the RWCS executes page advance processing. After the page heading report group, if defined, has been processed, the RWCS determines whether the saved next group integer location was set when the final body group was presented on the preceding page. Refer to rule 6 on page 14-40. If saved next group integer was not set, the body group shall be presented on the page to which the report is currently positioned. If saved next group integer was set, the RWCS moves the saved next group integer into LINE-COUNTER, resets saved next group integer to zero, and reapplies the previous paragraph of this rule.

- b. If a body group has been presented on the page to which the report is currently positioned, the RWCS computes a trial sum in a work location. The trial sum is computed by adding the content of LINE-COUNTER to the integers of all LINE NUMBER clauses of the report group. If the trial sum is not greater than the lower limit integer of the body group, then the report group is presented on the current page. If the trial sum exceeds the lower limit integer of the body group, then the RWCS executes page advance processing. After the page heading report group, if defined, has been processed, the RWCS reapplies the first part of this paragraph.

If no body group has yet been presented on the page to which the report is currently positioned, the RWCS determines whether the saved next group integer location was set when the final body group was presented on the preceding page. Refer to rule 6 on page 14-40.

If saved next group integer was not set, the body group is presented on the page to which the report is currently positioned.

If saved next group integer was set, the RWCS moves the saved next group integer into LINE-COUNTER, resets saved next group integer to zero, and computes a trial sum in a work location.

The trial sum is computed by adding the content of LINE-COUNTER to the integer one and the integers of all but the first LINE NUMBER clause of the body group. If the trial sum is not greater than the lower limit integer of the body group then the body group is presented on the current page. If the trial sum exceeds the lower limit integer of the body group, the RWCS executes page advance processing. After

the page heading report group (if defined) has been processed, the RWCS presents the body group on that page.

**		Applicable Rules ***							
		If the PAGE clause is specified.						If the PAGE clause is omitted.	
Sequence of LINE NUMBER clauses *	NEXT GROUP clause	Upper Limit	Lower Limit	Fit Test	First Print Line Position	Next Group	Final LINE-COUNTER Setting	First Print Line Position	Final LINE-COUNTER Setting
A R	Absolute	1	2	3a	4a	5	6a	Illegal Combination †	
A R	Relative	1	2	3a	4a		6b	Illegal Combination †	
A R	NEXT PAGE	1	2	3a	4a		6c	Illegal Combination †	
A R		1	2	3a	4a		6d	Illegal Combination †	
R	Absolute	1	2	3b	4b	5	6a	Illegal Combination ††	
R	Relative	1	2	3b	4b		6b	4d	6f
R	NEXT PAGE	1	2	3b	4b		6c	Illegal Combination ††	
R		1	2	3b	4d		6d	4d	6d
NP R	Absolute	1	2	3c	4a	5	6a	Illegal Combination †	
NP R	Relative	1	2	3c	4a		6b	Illegal Combination †	
NP R	NEXT PAGE	1	2	3c	4a		6c	Illegal Combination †	
NP R		1	2	3c	4a		6d	Illegal Combination	
					4c		6e	4d	6d

Figure 14-5. Body Group Presentation Rules

- * Refer to “LINE NUMBER Clause Notation” on page 14-34 for a description of the abbreviations used in column 1.
- ** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.
- *** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.
- † See “LINE NUMBER Clause” on page 14-49.
- †† See “NEXT GROUP Clause” on page 14-51.

-
- c. If a body group has been presented on the page to which the report is currently positioned, the RWCS executes page advance processing. After the page heading report group, if defined, has been processed, the RWCS reapplies this rule.

If no body group has yet been presented on the page to which the report is currently positioned, the RWCS determines whether the saved next group integer location was set when the final body group was presented on the preceding page. Refer to rule 6. If saved next group integer was not set, the body group shall be presented on the page to which the report is currently positioned. If saved next group integer was set, the RWCS moves the saved next group integer into LINE-COUNTER and resets saved next group integer to zero. If the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, the body group shall be presented on the page to which the report is currently positioned. Otherwise, the RWCS executes page advance processing. After the page heading report group (if defined) has been processed, the RWCS presents the body group on that page.

4. First print line position rules

- a. The first print line of the body group is presented on the line number specified by the integer of its LINE NUMBER clause.
- b. If the value in LINE-COUNTER is equal to or greater than the line number specified by the FIRST DETAIL phrase of the PAGE clause, and if no body group has previously been presented on the page to which the report is currently positioned, then the first print line of the current body group is presented on the line immediately following the line indicated by the value contained on LINE-COUNTER.

If the value in LINE-COUNTER is equal to or greater than the line number specified by the FIRST DETAIL phrase of the PAGE clause, and if a body group has previously been presented on the page to which the report is currently positioned, then the first print line of the current page group is presented on the line that is obtained by adding the content of LINE-COUNTER and the integer of the first LINE NUMBER clause of the current body group.

If the value in LINE-COUNTER is less than the line number specified by the FIRST DETAIL phrase of the PAGE clause, then the first printer line of the body group is presented on the line specified by the FIRST DETAIL phrase.

- c. The body group is not presented.
- d. The sum of the content of LINE-COUNTER and the integer of the first LINE NUMBER clause defines the line number on which the first print line is presented.

5. Next group rule

The integer of the absolute NEXT GROUP clause must specify a line number that is not less than that specified in the FIRST DETAIL phrase of the PAGE clause and that is not greater than that specified in the FOOTING phrase of the PAGE clause.

6. Final LINE-COUNTER setting rules

- a. If the body group that has just been presented is a control footing report group and if the control footing report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the control footing report group was presented.

For all other cases the RWCS makes a comparison of the line number on which the final print line of the body group was presented and the integer of the NEXT GROUP clause. If the former is less than the latter, then the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting. If the former is equal to or greater than the latter, then the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting; in addition, the RWCS places the NEXT GROUP integer into the saved next group integer location.

- b. If the body group that has just been presented is a control footing report group, and if the control footing report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the control footing report group was presented.

For all other cases the RWCS computes a trial sum in a work location. The trial sum is computed by adding the integer of the NEXT GROUP clause to the line number on which the final print line of the body group was presented. If the sum is less than the line number specified by the FOOTING phrase of the PAGE clause, then the RWCS places that sum into LINE-COUNTER as the final LINE-COUNTER setting. If the sum is equal to or greater than the line number specified by the FOOTING phrase of the PAGE clause, then the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting.

- c. If the body group that has just been presented is a control footing report group, and if the control footing report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the control footing report group was presented.

For all other cases the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting.

- d. The final LINE-COUNTER setting is the line number on which the final print line of the body group was presented.

- e. LINE-COUNTER is unaffected by the processing of a nonprintable body group.

- f. If the body group that has just been presented is a control footing report group, and if the control footing report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the control footing report group was presented.

For all other cases the RWCS places the sum of the line number on which the final print line was presented and the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

Page Footing Presentation Rules

Table 14-3 specifies the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a page footing report group.

**		Applicable Rules*** If the PAGE clause is specified.				
Sequence of LINE NUMBER clause*	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting
AR	Absolute	1	2	3a	4a	5a
AR	Relative	1	2	3a	4b	5b
AR		1	2	3a		5c
				3b		5d

* Refer to "LINE NUMBER Clause Notation" on page 14-34 for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

******* A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

Note: If the PAGE clause is omitted from the report description entry, then a page footing report group may not be defined. Refer to "TYPE Clause" on page 14-59.

The page footing presentation rules are as follows:

1. Upper limit rule

The first line number on which the page footing report group can be presented is the line number obtained by adding one to the value of integer 5 of the FOOTING phrase of the PAGE clause.

2. Lower limit rule

The last line number on which the page footing report group can be presented is the line number specified by integer 1 of the PAGE clause.

3. First print line position rules

a. The first print line of the page footing report group is presented on the line specified by the integer of its LINE NUMBER clause.

b. The page footing report group is not presented.

4. Next group rules

a. The NEXT GROUP integer must be greater than the line number on which the final print line of the page footing report group is presented. In addition, the NEXT GROUP integer must not be greater than the line number specified by integer 1 of the PAGE clause.

b. The sum of the NEXT GROUP integer and the line number on which the final print line of the page footing report group is presented must not be greater than the line number specified by integer 1 of the PAGE clause.

5. Final LINE-COUNTER setting rules

a. After the page footing report group is presented, the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

b. After the page footing report group is presented, the RWCS places the sum of the NEXT GROUP integer and the line number on which the final print line of the PAGE FOOTING report group was presented into LINE-COUNTER as the final LINE-COUNTER setting.

c. After the PAGE FOOTING report group is presented, the final LINE-COUNTER setting is the line number on which the final print line of the PAGE FOOTING report group was presented.

d. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

Report Footing Presentation Rules

Figure 14-6 on page 14-44 specifies the appropriate presentation rules for all permissible combinations of **LINE NUMBER** and **NEXT GROUP** clauses in a report footing report group. The report footing presentation rules are as follows:

1. Upper limit rules
 - a. If a page footing report group has been presented on the page to which the report is currently positioned, then the first line number on which the report footing report group can be presented is one greater than the final **LINE-COUNTER** setting established by the page footing report group.

Otherwise, the first line number on which the report footing report group can be presented is the line number obtained by adding one and the value of integer 5 of the **PAGE** clause.
 - b. The first line number on which the report footing report group can be presented is the line number specified by the **HEADING** phrase of the **PAGE** clause.
2. Lower limit rules
 - a. The last line number on which a control heading report group or detail report group can be presented is the line number specified by the last detail phrase of the **PAGE** clause.
 - b. The last line number on which a control footing report group can be presented is the line number specified by the **FOOTING** phrase of the **PAGE** clause.
3. First print line position
 - a. The first print line of the report footing report group is presented on the line specified by the integer of its **LINE NUMBER** clause.
 - b. If a page footing report group has been presented on the page to which the report is currently positioned, then the sum of the final **LINE-COUNTER** setting established by the page footing report group and the integer of the first **LINE NUMBER** clause of the report footing report group defines the line number on which the first print line of the report footing report group is presented. Otherwise, the sum of the integer of the first **LINE NUMBER** clause of the report footing report group and the line number specified by the value of integer-5 of the **FOOTING** phrase of the **PAGE** clause defines the line number on which the first print line of the report footing report group is presented.
 - c. The **NEXT PAGE** phrase in the first absolute **LINE NUMBER** clause directs that the report footing report group is presented on a page on which no other report group has been presented. The first print line of the report footing report group is presented on the line number specified by the integer of its **LINE NUMBER** clause.
 - d. The sum of the content of **LINE-COUNTER** and the integer of the first **LINE NUMBER** clause defines the line number on which the first print line is presented.
 - e. The report footing report group is not presented.
4. Final **LINE-COUNTER** setting rules
 - a. The final **LINE-COUNTER** setting is the line number on which the final print line of the report footing report group is presented.
 - b. **LINE-COUNTER** is unaffected by the processing of a nonprintable report group.

**		Applicable Rules ***						
		If the PAGE clause is specified.					If the PAGE clause is omitted.	
Sequence of LINE NUMBER clauses *	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting	First Print Line Position	Final LINE-COUNTER Setting
A R		1a	2	3a		4a	Illegal Combination †	
R		1a	2	3b		4a	3d	4a
N P R		1b	2	3c		4a	Illegal Combination †	
				3e		4b	3e	4b

Figure 14-6. Report Footing Presentation Rules

- * See "LINE NUMBER Clause" on page 14-49 for a description of the abbreviations used in column 1.
- ** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.
- *** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.
- † See "LINE NUMBER Clause" on page 14-49.

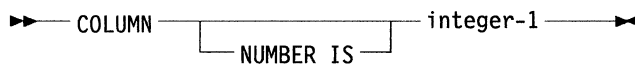
COLUMN NUMBER Clause

Function

The COLUMN NUMBER clause identifies a printable item and specifies the position of the item on a print line.

General Format

The following figure shows the format of the COLUMN NUMBER clause:



Syntax Rules

The following syntax rules apply to the COLUMN NUMBER clause:

1. The COLUMN NUMBER clause can be specified only at the elementary level within a report group. The COLUMN NUMBER clause, if present, must appear in or be subordinate to an entry that contains a LINE NUMBER clause.
2. Within a given print line, the printable items should be defined in ascending column number order such that each printable item defined occupies a unique sequence of contiguous character positions. *No restrictions on the sequence of COLUMN NUMBER clauses are enforced.*

OSVS

General Rules

The following rules apply to the COLUMN NUMBER clause:

1. The COLUMN NUMBER clause indicates that the object of a SOURCE clause, the object of a VALUE clause or the sum counter defined by a SUM clause is to be presented on the print line. The absence of a COLUMN NUMBER clause indicates that the entry is not to be presented on a print line.
2. integer-1 specifies the column number of the leftmost character position of the printable item.
3. The RWCS supplies space characters for all positions of a print line that are not occupied by printable items.
4. The leftmost position of the print line is considered to be column number 1.

Data-Name

Function

The data name specifies the name of the data-item being described.

General Format

The following figure shows the format of the data-name.

▶▶— data-name-1 —▶▶

Syntax Rule

In the REPORT SECTION, data-name-1 need not appear in a data description entry.

General Rules

In the REPORT SECTION, data-name-1 must be given in the following cases:

1. When data-name-1 represents a report group to be referred to by a GENERATE or a USE statement in the Procedure Division
2. When reference is to be made to the sum counter in the Procedure Division or REPORT SECTION
3. When a DETAIL report group is referenced in the UPON phrase of the SUM clause
4. When data-name-1 is required to provide sum counter qualification.

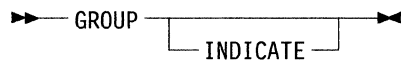
GROUP INDICATE Clause

Function

The **GROUP INDICATE** clause specifies that the associated printable item is presented only on the first occurrence of its report group after a control break or page advance.

General Format

The following figure shows the format of the **GROUP INDICATE** clause:



Syntax Rule

The **GROUP INDICATE** clause must only be specified in a **DETAIL** report group entry that defines a printable item.

General Rules

The following rules apply to the **GROUP INDICATE** clause:

1. If a **GROUP INDICATE** clause is specified, it causes the **SOURCE** or **VALUE** clause to be ignored and spaces supplied, except:
 - a. On the first presentation of the **DETAIL** report group in the report
 - b. On the first presentation of the **DETAIL** report group after every page advance
 - c. On the first presentation of the **DETAIL** report group after every control group.
2. If the report description entry specifies neither a **PAGE** clause nor a **CONTROL** clause, then a **GROUP INDICATE** printable item is presented the first time its **DETAIL** is presented after the **INITIATE** statement is executed. Thereafter, spaces are supplied for indicated items with **SOURCE** or **VALUE** clauses.

Level-Number

Function

The level-number indicates the position of a data-item within the hierarchical structure of a report group.

General Format

The following figure shows the format of the level-number.

▶▶ level-number ▶▶

Syntax Rules

The following syntax rules apply to the level-number:

1. A level-number is required as the first element in each data description entry.
2. Data description entries subordinate to an RD entry must have level-numbers 01 through 49 only.

General Rules

The following syntax rules apply to the level-number:

1. The level-number 01 identifies the first entry in a report group.
2. Multiple level 01 entries subordinate to a report description entry having the level indicator RD do not represent implicit redefinitions of the same area.

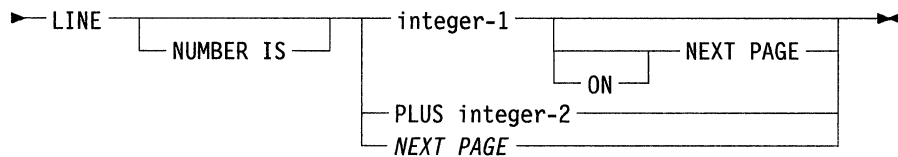
LINE NUMBER Clause

Function

The LINE NUMBER clause specifies vertical positioning information for its report group.

General Format

The following figure shows the format of the LINE NUMBER clause:



OSVS

Syntax Rules

The following syntax rules apply to the LINE NUMBER clause:

1. integer-1 and integer-2 must not exceed three significant digits in length.
Neither integer-1 nor integer-2 may be specified in such a way as to cause any line of a report group to be presented outside the vertical subdivision of the page designated for that report group type, as defined by the PAGE clause. Refer to “PAGE Clause” on page 14-22.
integer-2 may be zero.
2. Within a given report group description entry, an entry that contains a LINE NUMBER clause must not contain a subordinate entry that also contains a LINE NUMBER clause.
3. Within a given report group description entry, all absolute LINE NUMBER clauses must precede all relative LINE NUMBER clauses.
4. Within a given report group description entry, successive absolute LINE NUMBER clauses must specify integers that are in ascending order. The integers need not be consecutive.
5. If the PAGE clause is omitted from a given report description entry, only relative LINE NUMBER clauses may be specified in any report group description entry within that report.
6. Within a given report group description entry, a NEXT PAGE phrase may appear only once and, if present, must be in the first LINE NUMBER clause in that report group description entry.
7. A LINE NUMBER clause with the NEXT PAGE phrase may appear only in the description of body groups and in a REPORT FOOTING report group.
8. Every entry that defines a printable item (refer to “COLUMN NUMBER Clause” on page 14-45) must either contain a LINE NUMBER clause, or be subordinate to an entry that contains a LINE NUMBER clause.

-
9. The first **LINE NUMBER** clause specified within a **PAGE FOOTING** report group must be an absolute **LINE NUMBER** clause.

General Rules

The following general rules apply to the **LINE NUMBER** clause:

1. A **LINE NUMBER** clause must be specified to establish each print line of a report group.
2. The **RWCS** effects the vertical positioning specified by a **LINE NUMBER** clause before presenting the print line established by that **LINE NUMBER** clause.
3. **integer-1** specifies an absolute line number. An absolute line number specifies the line number on which the print line is presented.
4. **integer-2** specifies a relative line number. If a relative **LINE NUMBER** clause is not the first **LINE NUMBER** clause in the report group description entry, the line number on which its print line is presented is the line number on which the previous print line of the report group was presented, added to **integer-2** of the relative **LINE NUMBER** clause. If **integer-2** is zero, the line will be printed on the same line as the previous print line.

If a relative **LINE NUMBER** clause is the first **LINE NUMBER** clause in the report group description entry, then the line number on which its print line is presented is determined by specified rules. Refer to “Presentation Rules Tables” on page 14-32.

5. The **NEXT PAGE** phrase specifies that the report group is to be presented beginning on the indicated line number on a new page. Refer to “Presentation Rules Tables” on page 14-32.

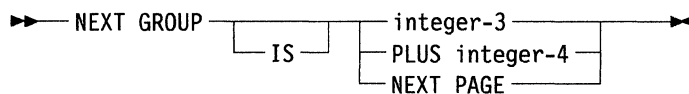
NEXT GROUP Clause

Function

The NEXT GROUP clause specifies information for vertical positioning of a page following the presentation of the last line of a report group.

General Format

The following figure shows the general format of the NEXT GROUP clause:



Syntax Rules

The following syntax rules apply to the NEXT GROUP clause:

1. A report group entry must not contain a NEXT GROUP clause unless the description of that report group contains at least one LINE NUMBER clause.
2. integer-3 and integer-4 must not exceed three significant digits in length.
3. If the PAGE clause is omitted from the report description entry, only a relative NEXT GROUP clause may be specified in any report group description entry within that report.
4. The NEXT PAGE phrase of the NEXT GROUP clause must not be specified in a PAGE FOOTING report group.
5. The NEXT GROUP clause must not be specified in a REPORT FOOTING report group or in a PAGE HEADING report group.

General Rules

The following general rules apply to the NEXT GROUP clause:

1. Any positioning of the page specified by the NEXT GROUP clause takes place after the presentation of the report group in which the clause appears. Refer to "Presentation Rules Tables" on page 14-32.
2. The RWCS uses the vertical positioning information supplied by the NEXT GROUP clause along with information from the TYPE and PAGE clauses and the value in LINE-COUNTER to determine a new value for LINE-COUNTER. Refer to "Presentation Rules Tables" on page 14-32.
3. The NEXT GROUP clause is ignored by the RWCS when it is specified on a CONTROL FOOTING report group that is at a level other than the highest level at which a control break is detected.

-
4. The NEXT GROUP clause of a body group refers to the next body group to be presented and, therefore, can affect the location at which the next body group is presented. The NEXT GROUP clause of a REPORT HEADING report group can affect the location at which the PAGE HEADING report group is presented. The NEXT GROUP clause of a PAGE FOOTING report group can affect the location at which the REPORT FOOTING report group is presented. Refer to “Presentation Rules Tables” on page 14-32.

SIGN Clause

Function

The SIGN clause specifies the position and the mode of representation of the operational sign when it is necessary to describe these properties explicitly.

General Format

The following figure shows the general format of the SIGN clause:



Syntax Rules

The following syntax rules apply to the SIGN clause:

1. The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character S.
2. The numeric data description entries to which the SIGN clause applies must be described, implicitly or explicitly, as USAGE IS DISPLAY.
3. When the SIGN clause is included in a report group description entry, the SEPARATE CHARACTER phrase must be specified.

General Rules

The following rules apply to the SIGN clause:

1. The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character S. The S indicates the presence of, but neither the representation nor, necessarily, the position of the operational sign.
2. A numeric data description entry, whose PICTURE contains the character S but to which no optional SIGN clause applies, has an operational sign, but neither the representation nor, necessarily, the position of the operational sign is specified by the character S. In this (default) case, rule 3 does not apply to such signed numeric data-items. The representation of the default operational sign is defined in "Selection of Character Representation and Radix" on page 2-20.
3. Since a SIGN clause in a report group description entry must specify the SEPARATE CHARACTER phrase:
 - a. The operational sign will be presumed to be the leading (or respectively, trailing) character position of the elementary numeric data-item. This character position is not a digit position.

-
- b. The letter S in a PICTURE character string is counted in determining the size of the item (in terms of standard data format characters).
 - c. The operational signs for positive and negative are the standard data format characters + and -, respectively.
 4. Each numeric data description entry whose PICTURE contains the character S is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

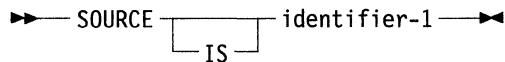
SOURCE Clause

Function

The SOURCE clause identifies the sending data-item that is moved to an associated printable item defined within a report group description entry.

General Format

The following figure shows the general format for the SOURCE clause:



Syntax Rules

The following rules apply to the SOURCE clause:

1. identifier-1 may be defined in any section of the Data Division. If identifier-1 is a REPORT SECTION item it must be a:
 - a. PAGE-COUNTER
 - b. LINE-COUNTER
 - c. Sum counter that is part of the report within which the SOURCE clause appears.
2. identifier-1 specifies the sending data-item of the implicit MOVE statement that the RWCS executes to move the content of the data-item referenced by identifier-1 to the printable item. identifier-1 must be defined so that it conforms to the rules for sending items in the MOVE statement. Refer to "MOVE Statement" on page 7-65.

General Rule

The RWCS formats the print lines of a report group just before presenting the report group. Refer to "TYPE Clause" on page 14-59. At this time the implicit MOVE statements specified by SOURCE clauses are executed by the RWCS.

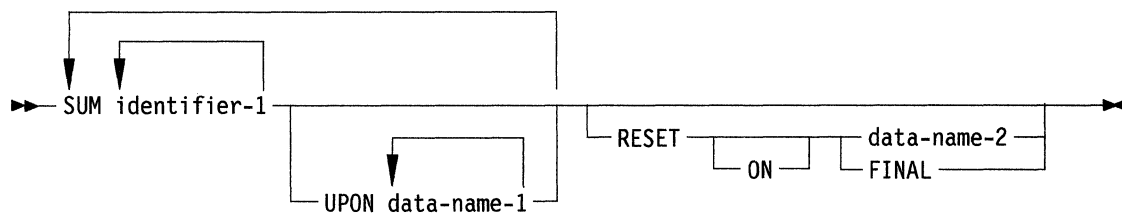
SUM Clause

Function

The SUM clause establishes a sum counter and names the data-items to be summed.

General Format

The following figure shows the general format of the SUM clause:



Syntax Rules

The following rules apply to the SUM clause:

1. identifier-1 must reference a numeric data-item. If identifier-1 is defined in the REPORT SECTION, identifier-1 must reference a sum counter.

If the UPON phrase is omitted, any identifiers in the associated SUM clause which are themselves sum counters must be defined either in the same report group that contains this SUM clause or in a report group which is at a lower level in the control hierarchy of this report.

If the UPON phrase is specified, any identifiers in the associated SUM clause must not be sum counters.

2. data-name-1 must be the name of a DETAIL report group described in the same report as the CONTROL FOOTING report group in which the SUM clause appears. data-name-1 may be qualified by a report name.
3. A SUM clause may appear only in the description of a CONTROL FOOTING report group.
4. data-name-2 must be one of the data names specified on the CONTROL clause for this report. data-name-2 must not be a lower level control than the associated control for the report group in which the RESET phrase appears.

FINAL, if specified in the RESET phrase, must also appear in the CONTROL clause for this report.

5. The highest permissible qualifier of a sum counter is the report name.

General Rules

The following rules apply to the SUM clause:

1. The SUM clause establishes a sum counter. The sum counter is a numeric data-item with an optional sign. At object time, the RWCS adds into the sum counter the value in each data-item referenced by identifier-1. This addition is consistent with the rules for arithmetic statements. Refer to "Arithmetic Statement Rules" on page 7-20.
2. The size of the sum counter is equal to the number of receiving character positions specified by the PICTURE clause that accompanies the SUM clause in the description of the elementary item.
3. Only one sum counter exists for an elementary report entry regardless of the number of SUM clauses specified in the elementary report entry.
4. If the elementary report entry for a printable item contains a SUM clause, the sum counter serves as a source data-item. The RWCS moves the data contained in the sum counter, according to the rules of the MOVE statement, to the printable item for presentation.
5. If a data-name appears as the subject of an elementary report entry that contains a SUM clause, the data name is the name of the sum counter; the data name is not the name of the printable item that the entry may also define.
6. It is permissible for Procedure Division statements to alter the contents of sum counters.
7. The RWCS adds data-item values referenced by identifiers into sum counters during the execution of GENERATE and TERMINATE statements. There are three categories of sum counter incrementing: subtotalling, crossfooting, and rolling forward. Subtotalling is accomplished only during execution of GENERATE statements and after any control break processing but before processing of the DETAIL report group. Refer to "GENERATE Statement" on page 14-68. Crossfooting and rolling forward are accomplished during the processing of CONTROL FOOTING report groups. Refer to "TYPE Clause" on page 14-59.
8. The UPON phrase provides the capability to accomplish selective subtotalling for the DETAIL report groups named in the phrase.
9. The RWCS adds each individual addend into the sum counter at a time that depends upon the characteristics of the addend.

- a. When the addend is a sum counter defined in the same CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed crossfooting.

Crossfooting occurs when a control break takes place and at the time the CONTROL FOOTING report group is processed.

Crossfooting is performed according to the sequence in which sum counters are defined within the CONTROL FOOTING report group. That is, all crossfooting into the first sum counter defined in the CONTROL FOOTING report group is completed, and then all crossfooting into the second sum counter defined in the CONTROL FOOTING report group is completed.

When one of the addends is the sum counter defined by the data description entry in which that SUM clause appears, the initial value of that sum counter at the time of summation is used in the summing operation.

- b. When the addend is a sum counter defined in a lower level CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed rolling forward. A sum counter in a lower level CONTROL FOOTING report group is rolled forward when a control break occurs and at the time that the lower level CONTROL FOOTING report group is processed.
- c. When the addend is not a sum counter, the accumulation into a sum counter of such an addend is called subtotalling. If the SUM clause contains the UPON phrase, the addends are subtotaled when a GENERATE statement for the desig-

nated **DETAIL** report group is executed. If the **SUM** clause does not contain the **UPON** phrase, the addends which are not sum counters are subtotaled when any **GENERATE** data name statement is executed for the report in which the **SUM** clause appears.

10. If two or more of the identifiers specify the same addend, then the addend is added into the sum counter as many times as the addend is referenced in the **SUM** clause. It is permissible for two or more of the data-names to specify the same **DETAIL** report group. When a **GENERATE** data name statement for such a **DETAIL** report group is given, the incrementing occurs repeatedly, as many times as data name appears in the **UPON** phrase.
11. The subtotaing that occurs when a **GENERATE** report name statement is executed is discussed in the appropriate paragraph. Refer to “**GENERATE** Statement” on page 14-68.
12. In the absence of an explicit **RESET** phrase, the **RWCS** will set a sum counter to zero at the time that the **RWCS** is processing the **CONTROL FOOTING** report group within which the sum counter is defined. If an explicit **RESET** phrase is specified, then the **RWCS** will set the sum counter to zero at the time that the **RWCS** is processing the designated level of the control hierarchy. Refer to the following section “**TYPE Clause**” on page 14-59.

Sum counters are initially set to zero by the **RWCS** during the execution of the **INITIATE** statement for the report containing the sum counter.

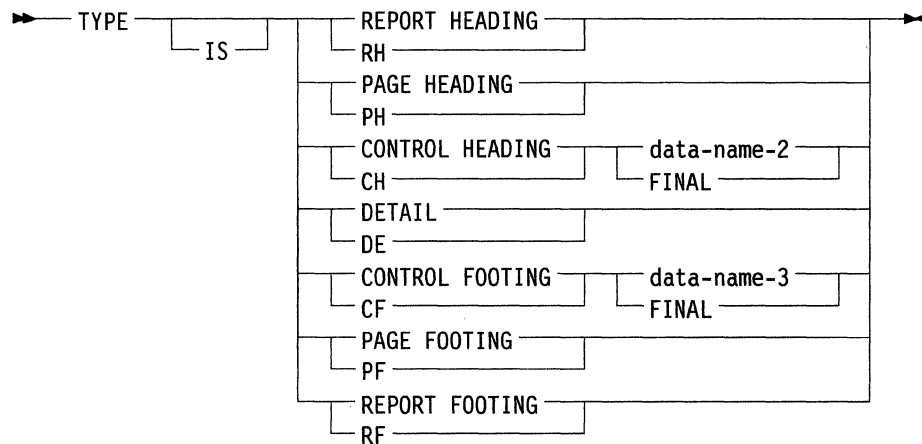
TYPE Clause

Function

The TYPE clause specifies the particular type of report group that is described by this entry and indicates the time at which the report group is to be processed by the RWCS.

General Format

The following figure shows the general format of the TYPE clause:



Syntax Rules

The following rules apply to the TYPE clause:

1. The following abbreviations are acceptable:
 - RH is an abbreviation for REPORT HEADING.
 - PH is an abbreviation for PAGE HEADING.
 - CH is an abbreviation for CONTROL HEADING.
 - DE is an abbreviation for DETAIL.
 - CF is an abbreviation for CONTROL FOOTING.
 - PF is an abbreviation for PAGE FOOTING.
 - RF is an abbreviation for REPORT FOOTING.
2. REPORT HEADING, PAGE HEADING, CONTROL HEADING FINAL, CONTROL FOOTING FINAL, PAGE FOOTING, and REPORT FOOTING report groups may each appear no more than once in the description of a report.
3. PAGE HEADING and PAGE FOOTING report groups may be specified only if a PAGE clause is specified in the corresponding report description entry.

-
4. data-name-2, data-name-3, and FINAL, if present, must be specified in the CONTROL clause of the corresponding report description entry. *This rule is not enforced with regard to FINAL.*

OSVS

At most, one CONTROL HEADING report group and one CONTROL FOOTING report group can be specified for each data name or FINAL in the CONTROL clause of the report description entry. However, neither a CONTROL HEADING report group nor a CONTROL FOOTING report group is required for a data name or FINAL specified in the CONTROL clause of the report description entry.

5. In CONTROL FOOTING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING report groups, SOURCE clauses and USE statements must not reference any of the following:
 - a. Group data-items containing a control data-item
 - b. Data-items subordinate to a control data-item
 - c. A redefinition or renaming of any part of a control data-item.

In PAGE HEADING and PAGE FOOTING report groups, SOURCE clauses and USE statements must not reference control data names.

6. When a GENERATE report name statement is specified in the Procedure Division, the corresponding report description entry must include no more than one DETAIL report group. If no GENERATE data name statements are specified for such a report, a DETAIL report group is not required.
7. The description of a report must include at least one body group.

General Rules

The following rules apply to the TYPE clause:

1. DETAIL report groups are processed by the RWCS as a direct result of GENERATE statements. If a report group is other than TYPE DETAIL, its processing is an automatic RWCS function.
2. The REPORT HEADING phrase specifies a report group that is processed by the RWCS only once each report, as the first report group of that report. The REPORT HEADING report group is processed during the execution of the chronologically first GENERATE statement for that report.
3. The PAGE HEADING phrase specifies a report group that is processed by the RWCS as the first report group on each page of that report except under the following conditions:
 - a. A PAGE HEADING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.
 - b. A PAGE HEADING report group is processed as the second report group on a page when it is preceded by a REPORT HEADING report group that is not to be presented on a page by itself.

Refer to "Presentation Rules Tables" on page 14-32.

4. The CONTROL HEADING phrase specifies a report group that is processed by the RWCS at the beginning of a control group for a designated control data name or, in the case of FINAL, is processed during the execution of the chronologically first GENERATE statement for that report. During the execution of any GENERATE statement at which the RWCS detects a control break, any CONTROL HEADING report groups associated with the highest control level of the break and lower levels are processed.
5. The DETAIL phrase specifies a report group that is processed by the RWCS when a corresponding GENERATE statement is executed.

-
6. The CONTROL FOOTING phrase specifies a report group that is processed by the RWCS at the end of a control group for a designated control data name.

In the case of FINAL, the CONTROL FOOTING report group is processed only once each report as the last body group of that report. During the execution of any GENERATE statement in which the RWCS detects a control break, any CONTROL FOOTING report group associated with the highest level of the control break or more minor levels is presented. All CONTROL FOOTING report groups are presented during the execution of the TERMINATE statement if there has been at least one GENERATE statement executed for the report. Refer to "TERMINATE Statement" on page 14-74.

7. The PAGE FOOTING phrase specifies a report group that is processed by the RWCS as the last report group on each page except under the following conditions:
 - a. A PAGE FOOTING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.
 - b. A PAGE FOOTING report group is processed as the second to last report group on a page when it is followed by a REPORT FOOTING report group that is not to be processed on a page by itself.

Refer to "Presentation Rules Tables" on page 14-32.

8. The REPORT FOOTING phrase specifies a report group that is processed by the RWCS only once each report and as the last report group of that report. The REPORT FOOTING report group is processed during execution of a corresponding TERMINATE statement if there has been at least one GENERATE statement executed for the report. Refer to "TERMINATE Statement" on page 14-74.
9. The sequence of steps that the RWCS executes when it processes a REPORT HEADING, PAGE HEADING, CONTROL HEADING, PAGE FOOTING, or REPORT FOOTING report group is described below.
 - a. If there is a USE BEFORE REPORTING procedure that references the data name of the report group, the USE procedure is executed.
 - b. If a SUPPRESS statement has been executed or if the report group is not printable, there is no further processing to be done for the report group.
 - c. If a SUPPRESS statement has not been executed and the report group is printable, the RWCS formats the print lines and presents the report group according to the presentation rules for that type of report group. Refer to "Presentation Rules Tables" on page 14-32.
10. The sequence of steps that the RWCS executes when it processes a CONTROL FOOTING report group is described below.

The GENERATE rules specify that when a control break occurs, the RWCS produces the CONTROL FOOTING report groups beginning at the minor level and proceeding upward through the level at which the highest control break was sensed. In this regard, it should be noted that even though no CONTROL FOOTING report group has been defined for a given control data name, the RWCS will still have to execute the step described in rule 10f on page 14-62 if a RESET phrase within the report description specifies that control data name.

- a. Sum counters are crossfooted; that is, all sum counters defined in this report group that are operands of SUM clauses in the same report group are added to their sum counters. Refer to "SUM Clause" on page 14-56.
- b. Sum counters are rolled forward; that is, all sum counters defined in the report group that are operands of SUM clauses in higher level CONTROL FOOTING report groups are added to the higher level sum counters. Refer to "SUM Clause" on page 14-56.
- c. A USE BEFORE REPORTING procedure references the data name of the report group, the USE procedure is executed.
- d. If a SUPPRESS statement has been executed or if the report group is not printable, the RWCS next executes the step described in rule 10f on page 14-62.

-
- e. If a SUPPRESS statement has not been executed and the report group is printable, the RWCS formats the print lines and presents the report group according to the presentation rules of CONTROL FOOTING report groups.
 - f. The report writer control system then resets those sum counters that are to be reset when the RWCS processes this level in the control hierarchy. Refer to "SUM Clause" on page 14-56.
11. The DETAIL report group processing what the RWCS executes in response to a GENERATE data name statement is described in rules 11a through 11e.

When the description of a report includes exactly one DETAIL report group, the detail-related processing that the RWCS executes in response to a GENERATE report name statement is described in rules 11a through 11d. These steps are performed as though a GENERATE data name statement were being executed.

When the description of a report includes no DETAIL report groups, the detail-related processing that the RWCS executes in response to a GENERATE report name statement is described in rule 11a. This step is performed as though the description of the report included exactly one DETAIL report group and as though GENERATE data name statement were being executed.

- a. The RWCS performs any subtotalling that has been designated for the DETAIL report group. Refer to "SUM Clause" on page 14-56.
 - b. If there is a USE BEFORE REPORTING procedure that refers to the data name of the report group, the USE procedure is executed.
 - c. If a SUPPRESS statement has been executed, or if the report group is not printable, there is no further processing done for the report group.
 - d. If the DETAIL report group is being processed as a consequence of a GENERATE report-name statement, there is no further processing done for the report group.
 - e. If neither 11c nor 11d applies, the RWCS formats the print lines and presents the report group according to the presentation rules for DETAIL report groups. Refer to "Presentation Rules Tables" on page 14-32.
12. When the RWCS is processing a CONTROL HEADING, CONTROL FOOTING, or DETAIL report group, as described in rules 9, 10, and 11, the RWCS may have to interrupt the processing of that body group after determining that the body group is to be presented and execute a page advance (and process PAGE FOOTING and PAGE HEADING report groups) before actually presenting the body group.
 13. During control break processing, the values of control data-items that the RWCS used to detect a given control break are referred to as prior values.
 - a. During control break processing of a CONTROL FOOTING report group, any references to control data-items in a USE procedure or SOURCE clause associated with that CONTROL FOOTING report group are supplied with prior values.
 - b. When a TERMINATE statement is executed, the RWCS makes the prior control data-item values available to SOURCE clause or USE procedure references in CONTROL FOOTING and REPORT FOOTING report groups as though a control break had been detected in the highest control data-name.
 - c. All other data-item references within report groups and their USE procedures access the current values that are contained within the data-items at the time the report group is processed.

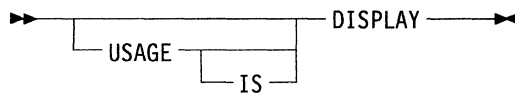
USAGE Clause

Function

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

General Format

The following figure shows the general format of the USAGE clause:



Syntax Rules

The following rules apply to the USAGE clause:

1. The USAGE clause may be written in any data description entry.
2. If the USAGE clause is written in the data description entry for a group item, it may also be written in the data description entry for a subordinate elementary item or group item.
3. The USAGE clause for a report group item can specify only USAGE IS DISPLAY.

General Rules

The following rules apply to the USAGE clause:

1. If the USAGE clause is written at a group level, it applies to each elementary item in the group.
2. The USAGE clause specifies the manner in which a data item is represented in the memory of a computer. It does not affect the use of the data item, although specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.
3. The USAGE IS DISPLAY clause indicates that the format of the data is a standard data format.
4. If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is implicitly DISPLAY.

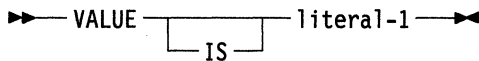
VALUE Clause

Function

The VALUE clause defines the value of REPORT SECTION printable items.

General Format

The following figure shows the general format of the VALUE clause:



Syntax Rules

The following rules apply to the VALUE clause:

1. A signed numeric literal must have a signed numeric PICTURE character string associated with it.
2. A numeric literal in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. A nonnumeric literal in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.
3. The VALUE clause must not be specified in any entry that is part of the description or redefinition of an external data record.

General Rules

The following rules apply to the VALUE clause:

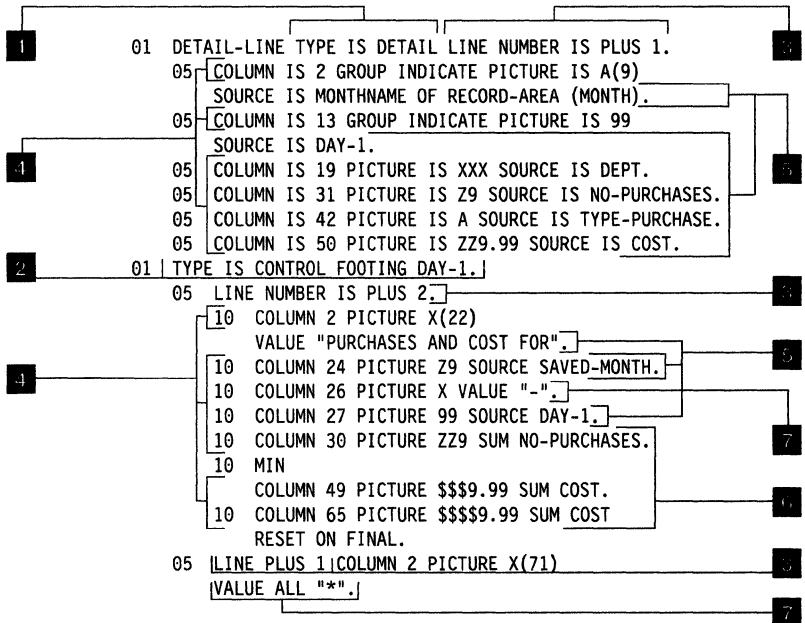
1. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item.
 - a. If the category of the item is numeric, literal-1 in the VALUE clause must be numeric.
 - b. If the category of the item is alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited, literal-1 in the VALUE clause must be a nonnumeric literal. The literal is aligned in the data item as if the data item had been described as alphanumeric. Refer to "Standard Alignment Rules" on page 2-19. Editing characters in the PICTURE clause are included in determining the size of the data item but have no effect on initialization of the data item. Refer to "PICTURE Clause" on page 6-18. Therefore, the value for an edited item must be specified in an edited form.
 - c. Initialization is not affected by any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.
2. In the REPORT SECTION, if the elementary report entry containing the VALUE clause does not contain a GROUP INDICATE clause, then the printable item will assume the specified value each time its report group is printed. However, when the GROUP INDICATE clause is also present, the specified value will be presented only

when certain object item conditions exist. Refer to "GROUP INDICATE Clause" on page 14-47.

Example

The following is an example of Report Group Description Entry and the resulting report lines.

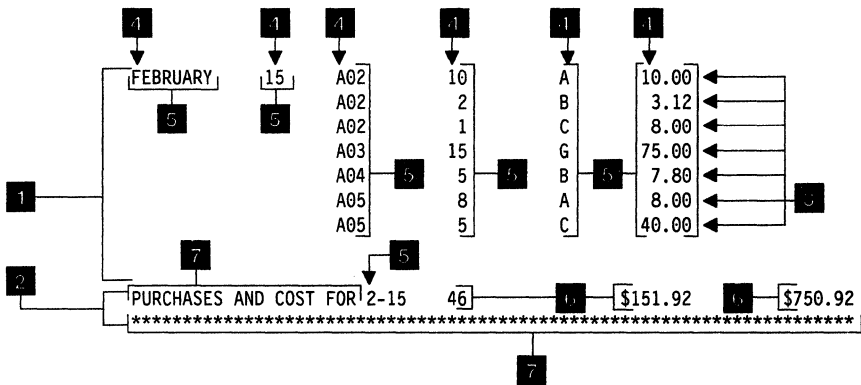
Report Writer Source Lines



Report Group Description Entry specifies:

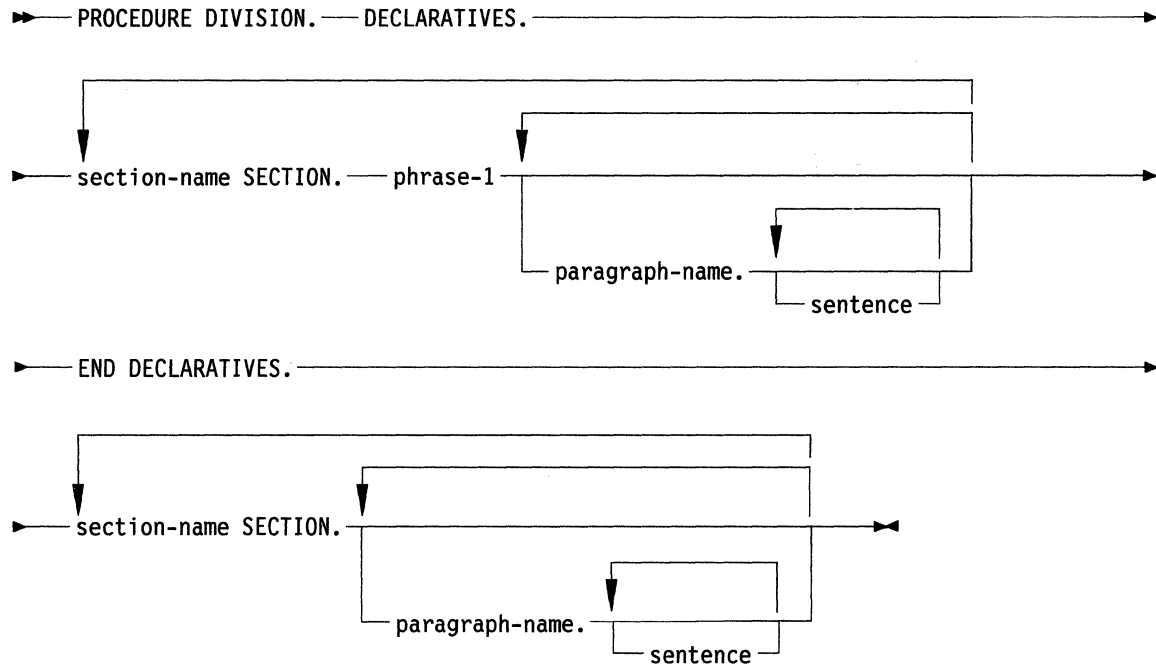
- FUNCTION - through TYPE clauses as:
 - TYPE DETAIL 1
 - TYPE CONTROL FOOTING 2
- VERTICAL SPACING - through relative LINE clauses 3
- HORIZONTAL SPACING - through COLUMN clauses 4
- CONTENTS - through:
 - SOURCE clauses 5
 - SUM clauses 6
 - VALUE clauses 7

Resulting Report Lines



Procedure Division in the Report Writer Module

The Procedure Division contains declarative procedures when the USE BEFORE REPORTING statement from the Report Writer module is present in a COBOL source program. Shown below is the general format of the Procedure Division when the USE BEFORE REPORTING statement is present.



where phrase-1 is:

▶ USE BEFORE REPORTING statement ▶

CLOSE Statement

Additional Syntax Rule

A report file must have sequential organization and open mode output. Refer to Chapter 8, "File Input and Output."

Additional General Rule

All reports associated with a report file that have been initiated must be ended with the execution of a **TERMINATE** statement before a **CLOSE** statement is executed for that report file.

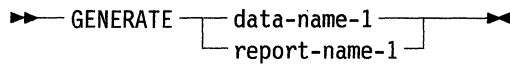
GENERATE Statement

Function

The GENERATE statement directs the RWCS to produce a report in accordance with the report description specified in the REPORT SECTION of the Data Division.

General Format

The following figure shows the general format of the GENERATE statement:



Syntax Rules

The following rules apply to the GENERATE statement:

1. data-name-1 must name a TYPE DETAIL report group and may be qualified by a report name.
2. report-name-1 may be used only if the referenced report description contains:
 - a. A CONTROL clause
 - b. Not more than one DETAIL REPORT GROUP
 - c. At least one body group.

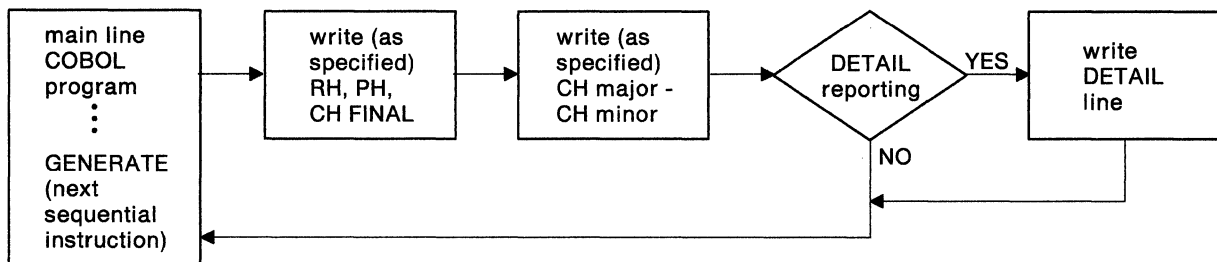
General Rules

The following rules apply to the GENERATE statement:

1. In response to a GENERATE report-name-1 statement, the RWCS performs summary processing. If all of the GENERATE statements that are executed for a report are of the form GENERATE report-name-1, then the report that is produced is called a summary report.
2. In response to a GENERATE data-name-1 statement, the RWCS performs detail processing that includes certain processing that is specific for the DETAIL report group designated by the GENERATE statement. Normally, the execution of a GENERATE data-name-1 statement causes the RWCS to present the designated DETAIL report group.
3. During the execution of the chronologically first GENERATE statement for a given report, the RWCS saves the values within the control data items. During the execution of the second and subsequent GENERATE statements for the same report, and until a control break is detected, the RWCS utilizes this set of control values to determine whether a control break has occurred. When a control break occurs, the RWCS saves the new set of control values, which it thereafter uses to sense for a control break until another control break occurs.

4. During the report presentation, an automatic function of the RWCS is to process PAGE HEADING and PAGE FOOTING report groups, if defined, when the RWCS must advance the report to a new page for the purpose of presenting a body group. Refer to "Presentation Rules Tables" on page 14-32.
5. When the chronologically first GENERATE statement for a given report is executed, the RWCS processes, in order, the report groups that are named below, provided that such report groups are defined within the report description. The RWCS also processes PAGE HEADING and PAGE FOOTING report groups as described in rule 4. The actions taken by the RWCS when it processes each type of report group are explained under the appropriate paragraph. Refer to "TYPE Clause" on page 14-59.
 - a. The REPORT HEADING report group is processed.
 - b. The PAGE HEADING report group is processed.
 - c. All CONTROL HEADING report groups are processed from major to minor.
 - d. If a GENERATE data-name-1 statement is being executed, the processing for the designated DETAIL report group is performed. If a GENERATE report-name-1 statement is being executed, a certain number of the steps that are involved in the processing of a DETAIL report group are performed. Refer to "TYPE Clause" on page 14-59.

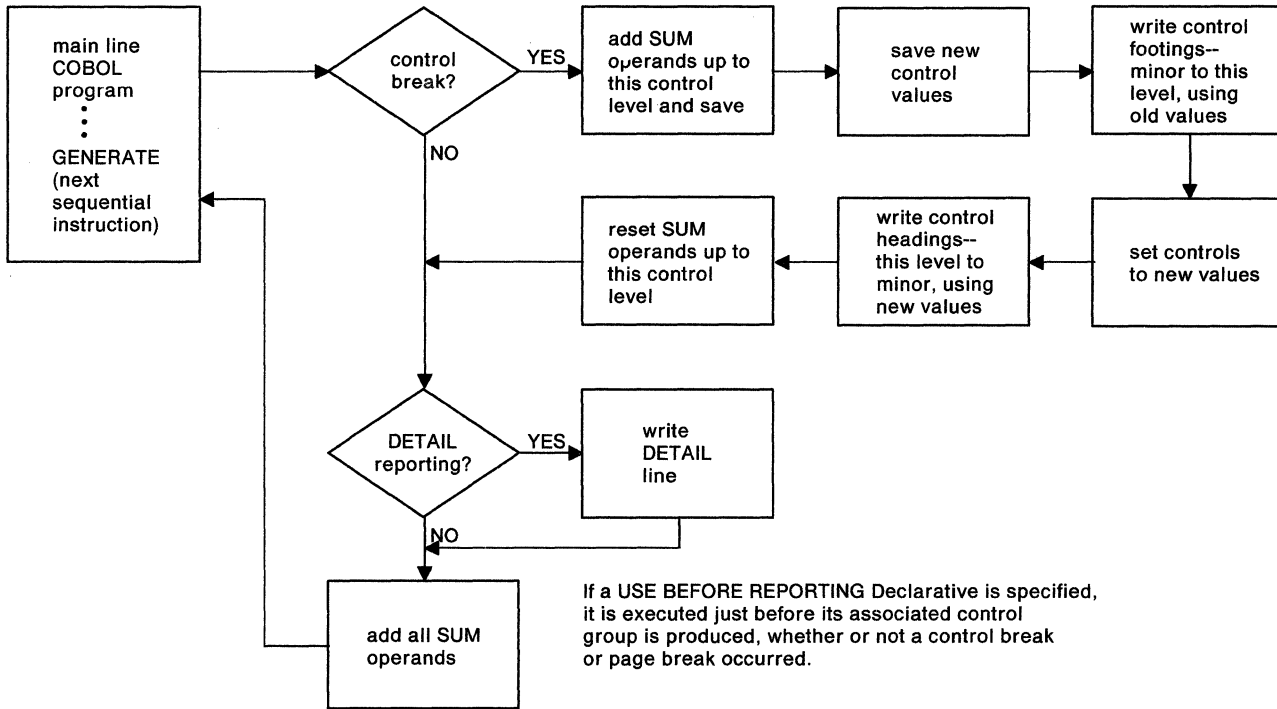
The following figure shows the sequence of operations of the FIRST GENERATE statement.



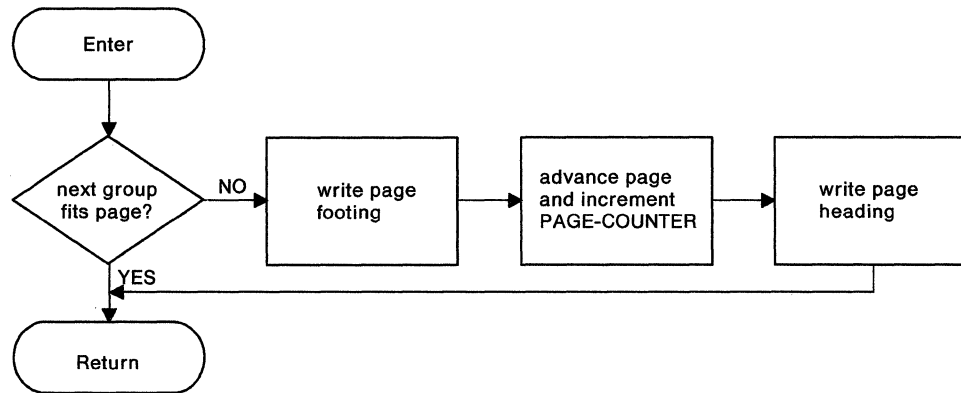
6. When a GENERATE statement other than the chronologically first is executed for a given report, the RWCS performs the steps listed below, as applicable. The RWCS also processes PAGE HEADING and PAGE FOOTING report groups as described in rule 4. The actions taken by the RWCS when it processes each type of report group are explained under the appropriate paragraph. Refer to "TYPE Clause" on page 14-59.
 - a. Sense for control break. The rules for determining the equality of control data items are the same as those specified for relation conditions. If a control break has occurred, then:
 - 1) Enable the CONTROL FOOTING USE procedures and CONTROL FOOTING SOURCE clauses to access the control data item values that the RWCS used to detect a given control break. Refer to "TYPE Clause" on page 14-59.
 - 2) Process the CONTROL FOOTING report groups in the order minor to major. Only CONTROL FOOTING report groups that are not more major than the highest level at which a control break occurred are processed.
 - 3) Process the CONTROL HEADING report groups in the order major to minor. Only the CONTROL HEADING report groups that are not more major than the highest level at which a control break occurred are processed.
 - b. If a GENERATE data-name-1 statement is being executed, the processing for the designated DETAIL report group is performed. If a GENERATE report-name-1 statement is being executed, certain of the steps that are involved in the processing of a DETAIL report group are performed. Refer to "TYPE Clause" on page 14-59.

7. **GENERATE** statements for a report can be executed only after an **INITIATE** statement for the report has been executed and before a **TERMINATE** statement for the report has been executed.

The following figure shows the sequence of operations of subsequent **GENERATE** statements.



Note: When the **PAGE** clause is specified, the following steps are executed before each printable body group is produced.



INITIATE Statement

Function

The INITIATE statement causes the RWCS to begin the processing of a report.

General Format

The following figure shows the general format of the INITIATE statement:



Syntax Rule

report-name-1 must be defined by a report description entry in the REPORT SECTION of the Data Division.

General Rules

The following rules apply to the INITIATE statement:

1. The INITIATE statement performs the following initialization functions for each named report:
 - a. All sum counters are set to zero
 - b. LINE-COUNTER is set to zero
 - c. PAGE-COUNTER is set to one.
2. The INITIATE statement does not place the file associated with the report in the open mode; therefore, an OPEN statement with either the OUTPUT phrase or the EXTEND phrase for the file must be executed before the execution of the INITIATE statement.
3. A subsequent INITIATE statement for report-name-1 must not be executed unless an intervening TERMINATE statement has been executed for report-name-1.
4. If more than one report name is specified in an INITIATE statement, the result of executing this INITIATE statement is the same as if a separate INITIATE statement had been written for each report name in the same order as specified in the INITIATE statement.

OPEN Statement

Additional Syntax Rules

The following rules apply to the OPEN statement:

1. A report file must have sequential organization.
2. The OPEN statement for a report file must contain only the OUTPUT phrase or the EXTEND phrase.

Refer to Chapter 8, "File Input and Output."

Additional General Rule

The OPEN statement for a report file must be executed prior to the execution of an INITIATE statement for any reports contained in the file.

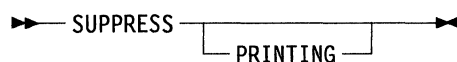
SUPPRESS Statement

Function

The SUPPRESS statement causes the RWCS to inhibit the presentation of a report group.

General Format

The following figure shows the general format of the SUPPRESS statement:



Syntax Rule

The SUPPRESS statement may only appear in a USE BEFORE REPORTING procedure.

General Rules

The following rules apply to the SUPPRESS statement:

1. The SUPPRESS statement inhibits presentation only for the report group named in the USE procedure within which the SUPPRESS statement appears.
2. The SUPPRESS statement must be executed each time the presentation of the report group is to be inhibited.
3. When the SUPPRESS statement is executed, the RWCS is instructed to inhibit the processing of the following report group functions:
 - a. The presentation of the print lines of the report group.
 - b. The processing of all LINE clauses in the report group.
 - c. The processing of the NEXT GROUP clause in the report group.
 - d. The adjustment of LINE-COUNTER.
4. *The SUPPRESS PRINTING function may also be achieved by moving OSVS the value 1 to the special register PRINT-SWITCH.*

The statement:

MOVE 1 TO PRINT-SWITCH

has exactly the same effect as the statement:

SUPPRESS PRINTING

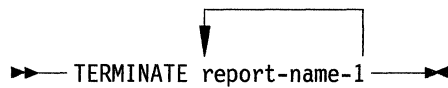
TERMINATE Statement

Function

The **TERMINATE** statement causes the RWCS to complete the processing of the specified reports.

General Format

The following figure shows the general format of the **TERMINATE** statement:



Syntax Rule

report-name-1 must be defined by a report description entry in the **REPORT SECTION** of the Data Division.

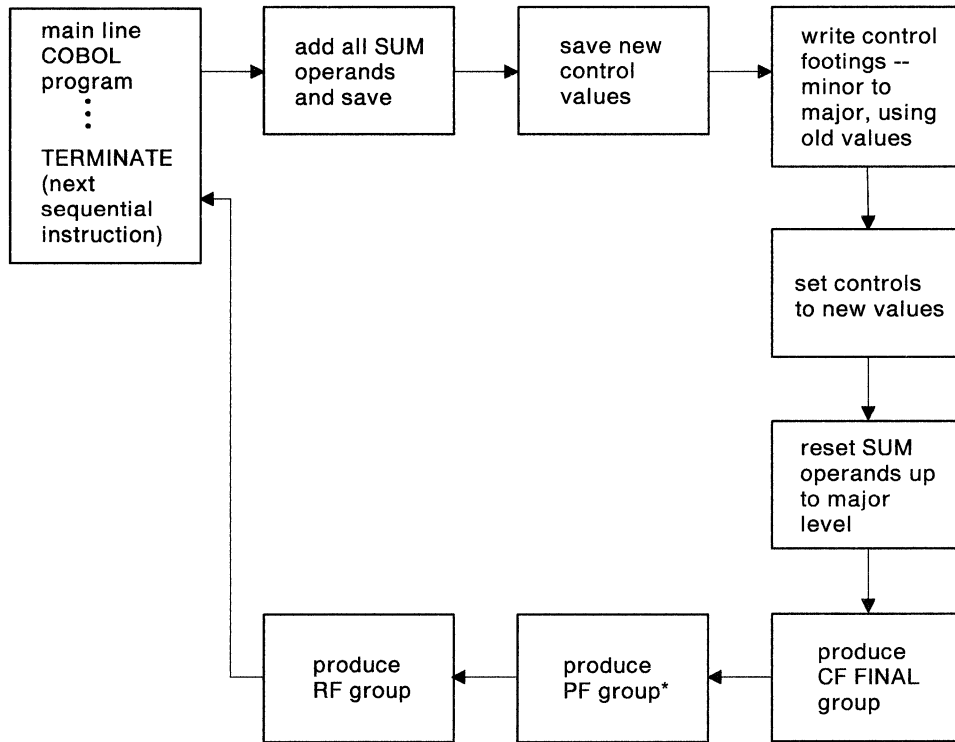
General Rules

The following rules apply to the **TERMINATE** statement:

1. The **TERMINATE** statement causes the RWCS to produce all the **CONTROL FOOTING** report groups beginning with the minor **CONTROL FOOTING** report group. The **REPORT FOOTING** report group is then produced. The RWCS makes the prior set of control data item values available to the **CONTROL FOOTING** and **REPORT FOOTING SOURCE** clauses and **USE** procedures, as though a control break had been sensed in the most major control data-name.
2. If no **GENERATE** statements have been executed for a report between the execution of an **INITIATE** statement and a **TERMINATE** statement, the **TERMINATE** statement does not cause the RWCS to produce any report groups or perform any of the related processing for that report.
3. During report presentation, an automatic function of the RWCS is to process **PAGE HEADING** and **PAGE FOOTING** report groups, if defined, when the RWCS must advance the report to a new page for presenting a body group. Refer to "Presentation Rules Tables" on page 14-32.
4. The **TERMINATE** statement cannot be executed for a report unless the **TERMINATE** statement was chronologically preceded by an **INITIATE** statement for which no **TERMINATE** statement has yet been executed.
5. If more than one report name is specified in a **TERMINATE** statement, the result of executing this **TERMINATE** statement is the same as if a separate **TERMINATE** statement had been written for each report name in the same order as specified in the **TERMINATE** statement.

6. The **TERMINATE** statement does not close the file with which the report is associated; a **CLOSE** statement for the file must be executed. Every report that is in an initiated condition must be terminated before a **CLOSE** statement is executed for the associated file.

The figure below shows the sequence of operations of the **TERMINATE** statement.



*- omitted if there is
no PAGE clause

USE BEFORE REPORTING Statement

Function

The **USE BEFORE REPORTING** statement specifies Procedure Division statements that are executed just before a report group named in the **REPORT SECTION** of the Data Division is presented.

General Format

The following figure shows the general format of the **USE BEFORE REPORTING** statement:

►— **USE BEFORE REPORTING identifier-1** —►

Syntax Rules

The following rules apply to the **USE BEFORE REPORTING** statement:

1. A **USE BEFORE REPORTING** statement, when present, must immediately follow a section header in the declaratives portion of the Procedure Division and must appear in a sentence by itself. The remainder of the section must consist of zero, one, or more procedural paragraphs that define the procedures to be used.
2. identifier-1 must reference a report group. identifier-1 must not appear in more than one **USE BEFORE REPORTING** statement.
3. The **GENERATE**, **INITIATE**, or **TERMINATE** statements must not appear in a paragraph within a **USE BEFORE REPORTING** procedure. A **PERFORM** statement in a **USE BEFORE REPORTING** procedure must not have **GENERATE**, **INITIATE**, or **TERMINATE** statements in its range.
4. A **USE BEFORE REPORTING** procedure must not alter the value of any control data item.
5. The **USE BEFORE REPORTING** statement itself is never executed; it merely defines the conditions calling for the execution of the **USE** procedures.

General Rules

The following rules apply to the **USE BEFORE REPORTING** statement:

1. A declarative procedure is invoked just before the named report group is produced during the execution of the program. The report group is named by identifier-1 in the **USE BEFORE REPORTING** statement which prefaces the declarative.
2. Within a declarative procedure, there must be no reference to any nondeclarative procedures.
3. Procedure names associated with a **USE BEFORE REPORTING** statement may be referenced in a different declarative section or in a nondeclarative procedure only with a **PERFORM** statement.

-
4. In the USE BEFORE REPORTING statement, the designated procedures are executed by the RWCS just before the named report group is produced. Refer to "TYPE Clause" on page 14-59.
 5. Within a USE procedure, there must not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

Report Writer Sample Program

The following example shows a complete Report Writer sample program.

This program is run after the close of the last working day of each month of the year. It supplies a record of purchases made by the Acme Manufacturing Company during the current year. Detail purchases for the current month are reported. For previous months, only a summary of each day's purchases is reported. In addition, a summary of the total number and amount of purchases is presented for each month, each quarter, and for the year to date.

INFILE, the input file, is an indexed file of the current year's purchases ordered on the invoice number of each purchase, with alternate ordering both on the date of purchase and on department. (This same file is used for report programs other than the one illustrated here.) INFILE is created via remote terminal from the comptroller's office; after each purchase order is signed by the comptroller, details of the purchase are added to the file from the terminal.

REPORT-FILE, the output file, is a physical sequential file assigned to the printer.

IDENTIFICATION DIVISION.

PROGRAM-ID. ACME.

REMARKS. THE REPORT WAS PRODUCED BY THE REPORT WRITER.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-RISC-6000.

OBJECT-COMPUTER. IBM-RISC-6000.

SPECIAL-NAMES.

SWITCH 0 IS SW ON IS YEAR-END.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT INFILE ASSIGN TO INXMAS1

ORGANIZATION IS INDEXED

ACCESS IS DYNAMIC

RECORD KEY IS ORDER-NO PASSWORD IS DEPT-PASS

ALTERNATE RECORD KEY IS DATE-PURCHASE

WITH DUPLICATES

ALTERNATE RECORD KEY IS DEPT

WITH DUPLICATES

FILE STATUS IS SK.

SELECT REPORT-FILE ASSIGN TO "REPORTFL".

DATA DIVISION.

FILE SECTION.

FD INFILE

LABEL RECORDS ARE OMITTED

DATA RECORD IS INPUT-RECORD.

01 INPUT-RECORD.

05 ORDER-NO PIC X(6).

05 DEPT PIC XXX.

05 DATE-PURCHASE.

10 MM PIC 99.

10 DD PIC 99.

10 YY PIC 99.

05 NO-PURCHASES PIC 99.

05 TYPE-PURCHASE PIC A.

05 COST PIC 999V99.

FD REPORT-FILE

LABEL RECORDS ARE OMITTED

RECORD CONTAINS 121 CHARACTERS

REPORT IS EXPENSE-REPORT.

WORKING-STORAGE SECTION.

77 SAVED-MONTH PIC 99 VALUE IS 0.
77 CONTINUED PIC X(11) VALUE IS SPACE.
77 QUARTER-USED PIC X(7).
77 DEPT-PASS PIC X(8) VALUE SPACES.
77 QUARTER PIC 9 VALUE 0.
77 REM PIC 9.
77 SK PIC XX.
01 TODAYS-DATE.
05 YR PIC 99.
05 MO PIC 99.
05 DA PIC 999.
01 QUARTER-NAMES VALUE "1ST2ND3RD4TH"
05 QUARTERNAME PIC X(3) OCCURS 4.
01 DATE-USED.
05 THIS-CENT PIC XX.
05 THIS-YEAR PIC XX.
05 FILLER PIC XXX VALUE SPACES.
01 MONTH-TABLE-1.
05 RECORD-MONTH.
10 FILLER PICTURE A(9) VALUE IS "JANUARY ".
10 FILLER PICTURE A(9) VALUE IS "FEBRUARY ".
10 FILLER PICTURE A(9) VALUE IS "MARCH ".
10 FILLER PICTURE A(9) VALUE IS "APRIL ".
10 FILLER PICTURE A(9) VALUE IS "MAY ".
10 FILLER PICTURE A(9) VALUE IS "JUNE ".
10 FILLER PICTURE A(9) VALUE IS "JULY ".
10 FILLER PICTURE A(9) VALUE IS "AUGUST ".
10 FILLER PICTURE A(9) VALUE IS "SEPTEMBER".
10 FILLER PICTURE A(9) VALUE IS "OCTOBER ".
10 FILLER PICTURE A(9) VALUE IS "NOVEMBER ".
10 FILLER PICTURE A(9) VALUE IS "DECEMBER ".
05 RECORD-AREA REDEFINES RECORD-MONTH.
10 MONTHNAME PICTURE A(9) OCCURS 12 TIMES.

REPORT SECTION.

RD EXPENSE-REPORT
 CONTROLS ARE FINAL QUARTER MM DD
 PAGE LIMIT IS 59 LINES
 HEADING 1
 FIRST DETAIL 9
 LAST DETAIL 48
 FOOTING 52.

01 TYPE IS REPORT HEADING.
 05 LINE NUMBER IS 1
 COLUMN NUMBER IS 27
 PICTURE IS A(26)
 VALUE IS "ACME MANUFACTURING COMPANY".
 05 LINE NUMBER IS 3
 COLUMN NUMBER IS 26
 PICTURE IS A(27)
 VALUE IS "RUNNING EXPENDITURES REPORT".

01 PAGE-HEAD
 TYPE IS PAGE HEADING.
 05 LINE NUMBER IS 5.
 10 COLUMN IS 30
 PICTURE IS A(9)
 SOURCE IS MONTHNAME (MM).
 10 COLUMN IS 39
 PICTURE IS A(12)
 VALUE IS "EXPENDITURES".
 10 COLUMN IS 52
 PICTURE IS X(11)
 SOURCE IS CONTINUED.

05 LINE IS 7.
 10 COLUMN IS 2
 PICTURE IS X(35)
 VALUE IS "MONTH DAY DEPT NO-PURCHASES".
 10 COLUMN IS 40
 PICTURE IS X(33)
 VALUE IS "TYPE COST CUMULATIVE-COST".

01 DETAIL-LINE TYPE IS DETAIL LINE NUMBER IS PLUS 1.
 05 COLUMN IS 2 GROUP INDICATE PICTURE IS A(9)
 SOURCE IS MONTHNAME (MM).
 05 COLUMN IS 13 GROUP INDICATE PICTURE IS 99
 SOURCE IS DD.
 05 COLUMN IS 19 PICTURE IS XXX SOURCE IS DEPT.
 05 COLUMN IS 31 PICTURE IS Z9 SOURCE IS NO-PURCHASES.
 05 COLUMN IS 42 PICTURE IS A SOURCE IS TYPE-PURCHASE.
 05 COLUMN IS 50 PICTURE IS ZZ9.99 SOURCE IS COST.

```

01 TYPE IS CONTROL FOOTING DD.
05 LINE NUMBER IS PLUS 2.
10 COLUMN 2 PICTURE X(22)
   VALUE "PURCHASES AND COST FOR".
10 COLUMN 24 PICTURE Z9 SOURCE SAVED-MONTH.
10 COLUMN 26 PICTURE X VALUE "-".
10 COLUMN 27 PICTURE 99 SOURCE DD.
10 COLUMN 30 PICTURE ZZ9 SUM NO-PURCHASES.
10 MIN
   COLUMN 49 PICTURE 9.99 SUM COST.
10 COLUMN 65 PICTURE $9.99 SUM COST
   RESET ON FINAL.
05 LINE PLUS 1 COLUMN 2 PICTURE X(71)
   VALUE ALL "*".
01 TYPE CONTROL FOOTING MM
LINE PLUS 1 NEXT GROUP NEXT PAGE.
05 COLUMN 16 PICTURE A(14) VALUE "TOTAL COST FOR".
05 COLUMN 31 PICTURE A(9)
   SOURCE MONTHNAME (MM).
05 COLUMN 43 PICTURE AAA VALUE "WAS".
05 INT
   COLUMN 48 PICTURE 9.99 SUM MIN.
01 QUARTER-FOOT TYPE CONTROL FOOTING QUARTER
LINE NEXT PAGE.
05 COLUMN 16 PIC A(14)
   VALUE "TOTAL COST FOR".
05 COLUMN 31 PIC X(3)
   SOURCE QUARTERNAME (QUARTER).
05 COLUMN 35 PIC A(7)
   VALUE "QUARTER".
05 COLUMN 43 PIC A(7)
   SOURCE QUARTER-USED.
05 QUARTER-INT COLUMN 52 PIC $9.99
   SUM INT.
01 FINAL-FOOT TYPE CONTROL FOOTING FINAL
LINE NEXT PAGE.
05 COLUMN 16 PIC A(19)
   VALUE "TOTAL COST FOR YEAR".
05 COLUMN 36 PIC A(7)
   SOURCE DATE-USED.
05 COLUMN 52 PIC $$9.99
   SUM QUARTER-INT.
01 TYPE PAGE FOOTING LINE 57.
05 COLUMN 59 PICTURE X(12) VALUE "REPORT-PAGE-".
05 COLUMN 71 PICTURE 99 SOURCE PAGE-COUNTER.
01 TYPE REPORT FOOTING
LINE PLUS 1 COLUMN 32 PICTURE A(13)
VALUE "END OF REPORT".

```

PROCEDURE DIVISION.
DECLARATIVES.
PAGE-HEAD-RTN SECTION.
 USE BEFORE REPORTING PAGE-HEAD.
PAGE-HEAD-RTN-TEST.
 IF MM NOT = SAVED-MONTH NEXT SENTENCE
 ELSE MOVE "(CONTINUED)" TO CONTINUED
 GO TO PAGE-HEAD-RTN-EXIT.
 IF SAVED-MONTH = 3 OR 6 OR 9 OR YEAR-END
 MOVE 1 TO PRINT-SWITCH MOVE 0 TO REM
 ELSE MOVE SPACES TO CONTINUED MOVE 1 TO REM.
 MOVE MM TO SAVED-MONTH.
PAGE-HEAD-RTN-EXIT.
 EXIT.
QUARTER-FOOT-ROUTINE SECTION.
 USE BEFORE REPORTING QUARTER-FOOT.
QUARTER-FOOT-PROCESS.
 IF YEAR-END MOVE 4 TO QUARTER.
 IF REM NOT = 0 MOVE "TO DATE" TO QUARTER-USED
 ELSE MOVE "WAS " TO QUARTER-USED.
QUARTER-FOOT-PROCESS-END.
 EXIT.
FINAL-FOOT-ROUTINE SECTION.
 USE BEFORE REPORTING FINAL-FOOT.
FINAL-FOOT-PROCESS.
 IF YEAR-END
 MOVE "19" TO THIS-CENT
 MOVE YY TO THIS-YEAR
 ELSE MOVE "TO DATE" TO DATE-USED.
FINAL-FOOT-PROCESS-END.
 EXIT.
END DECLARATIVES.

```
NONDECLARATIVES SECTION.
BEGIN-PROCESS.
  ACCEPT DEPT-PASS.
  ACCEPT TODAYS-DATE FROM DATE.
  OPEN INPUT INFILE OUTPUT REPORT-FILE.
  IF SK NOT = "00" GO TO EXCEPTION-2.
  MOVE 010101 TO DATE-PURCHASE.
  START INFILE KEY NOT LESS THAN DATE-PURCHASE.
  INITIATE EXPENSE-REPORT.
READATA.
  READ INFILE NEXT.
  IF SK NOT = "00" GO TO EXCEPTION-1.
  IF MM NOT = SAVED-MONTH
    PERFORM QUARTER-CALC-1 THRU QUARTER-CALC-END
    ELSE GO TO SWITCH-PARA.
  IF MM = MO
    ALTER SWITCH-PARA TO PROCEED TO DETAIL-REPORT.
SWITCH-PARA.
  GO TO SUMMARY-REPORT.
SUMMARY-REPORT.
  GENERATE EXPENSE-REPORT.
  GO TO READATA.
DETAIL-REPORT.
  GENERATE DETAIL-LINE.
  GO TO READATA.
QUARTER-CALC-1.
  IF SAVED-MONTH = 3 OR 6 OR 9 OR YEAR-END
    ADD 1 TO QUARTER.
QUARTER-CALC-END.
  EXIT.
EXCEPTION-1.
  TERMINATE EXPENSE-REPORT.
  IF SK = "10" GO TO END-PROGRAM.
  DISPLAY "INPUT RECORD = " INPUT-RECORD.
EXCEPTION-2.
  DISPLAY "STATUS KEY = " SK.
  DISPLAY "SAVE CONSOLE SHEET".
END-PROGRAM.
  CLOSE INFILE REPORT-FILE.
  STOP RUN.
```

Chapter 15. Communication

Contents

About This Chapter	15-3
Introduction	15-4
Data Division in the Communication Module	15-4
COMMUNICATION SECTION	15-4
Communication Description — Complete Entry Skeleton	15-4
Example 1	15-16
Example 2	15-16
Example 3	15-17
Example 4	15-17
Example 5	15-17
Example 6	15-17
Example 7	15-18
Example 8	15-18
Procedure Division in the Communication Module	15-19
ACCEPT MESSAGE COUNT Statement	15-20
Function	15-20
General Format	15-20
Syntax Rule	15-20
General Rules	15-20
DISABLE Statement	15-21
Function	15-21
General Format	15-21
Syntax Rules	15-21
General Rules	15-21
ENABLE Statement	15-23
Function	15-23
General Format	15-23
Syntax Rules	15-23
General Rules	15-23
PURGE Statement	15-25
Function	15-25
General Format	15-25
Syntax Rule	15-25
General Rules	15-25
RECEIVE Statement	15-26
Function	15-26
General Format	15-26
Syntax Rule	15-26
General Rules	15-26
SEND Statement	15-29
Function	15-29
General Format	15-29
Syntax Rules	15-29
General Rules	15-30
Communication Sample Program	15-33

About This Chapter

This chapter describes the AIX VS COBOL Communications module and how it provides the ability to access, process, and create messages among various communication programs and devices.

AIX VS COBOL accepts the syntax for the ANSI Communications optional module. However, Communications is not supported at run time. Therefore, you can compile programs that use this Communications syntax, but you cannot run them on this system.

Introduction

The communication module provides the ability to access, process, and create messages or portions thereof. It provides the ability to communicate through a Message Control System (MCS) with local and remote communication devices.

Data Division in the Communication Module

Data Division in the Communication Module contains two sections: the COMMUNICATION SECTION and the Communication Description — Complete Entry Skeleton.

COMMUNICATION SECTION

In a COBOL program the communication description (CD) entries represent the highest level of organization in the COMMUNICATION SECTION. The COMMUNICATION SECTION header is followed by a communication description entry consisting of a level indicator (CD), a data-name and a series of independent clauses. These clauses indicate the queues and subqueues, the message data and time, the source, the text length, the status and end keys, and message count of input. These clauses specify the destination count, the text length, the status and error keys, and destinations for output. For an input-output communication description entry the clauses specify the message date, message time, symbolic terminal, text length, end key, and status key. The entry itself is terminated by a period. These record areas may be implicitly redefined by user-specified record description entries following the various communication description clauses.

Communication Description — Complete Entry Skeleton

This section defines the function and the general rules of the Communication Description - Complete Skeleton Entry.

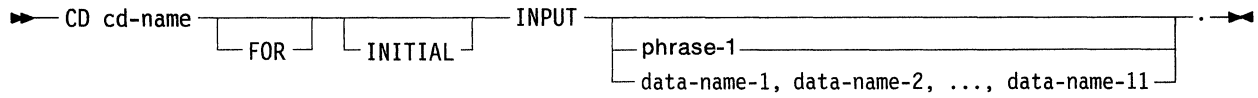
Function

The communication description specifies the interface area between the MCS and a COBOL program.

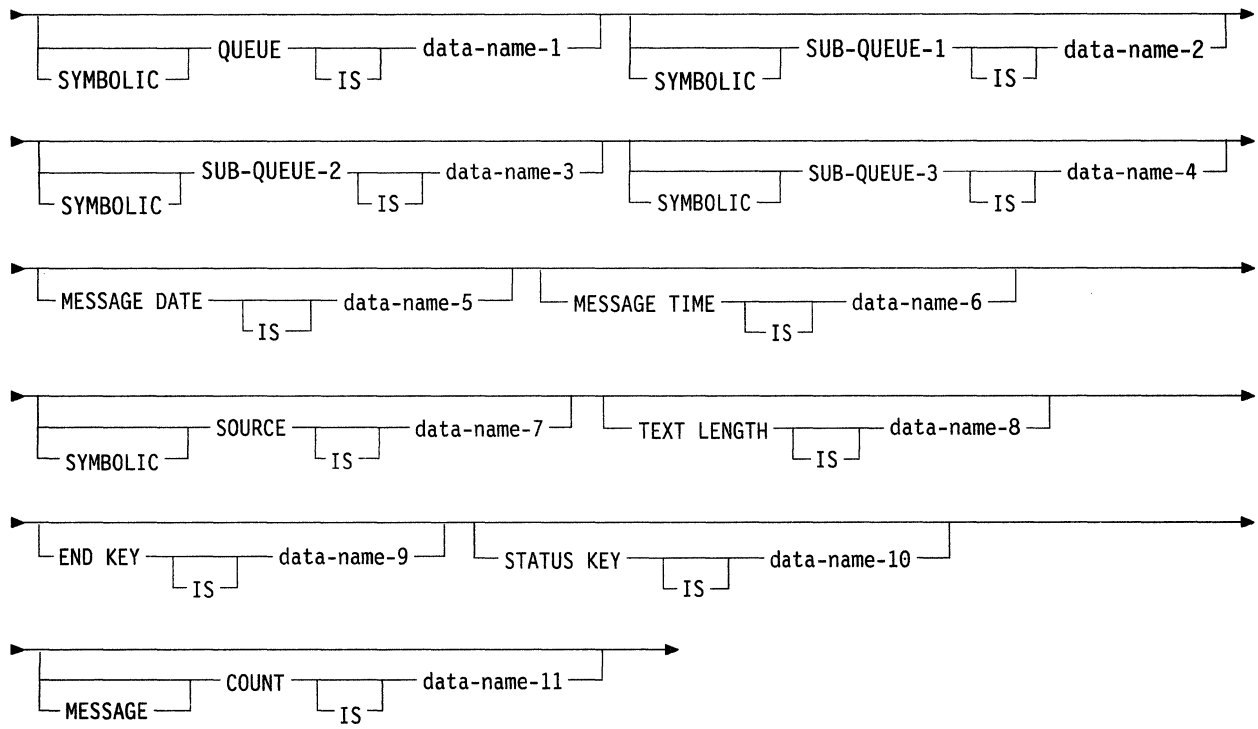
General Format

The following figures show the format of the communication description:

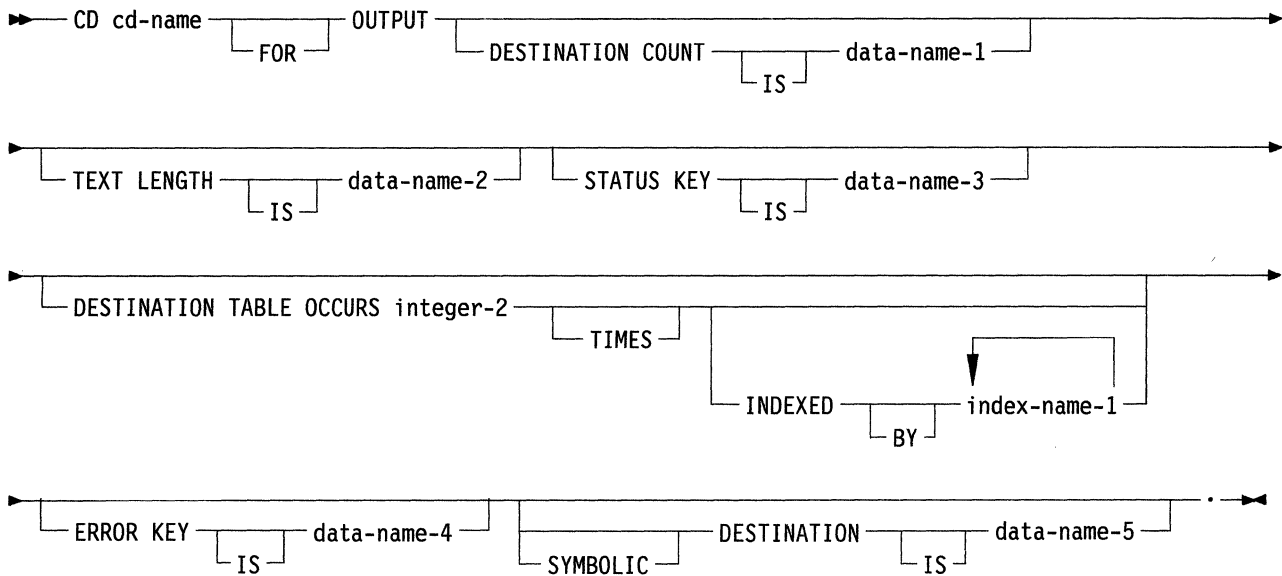
Format 1



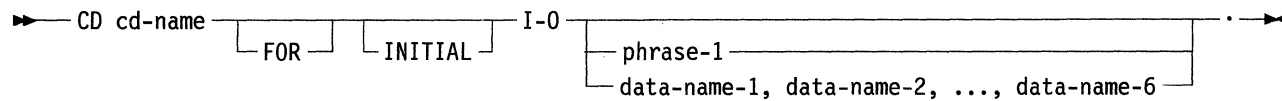
where phrase-1 is:



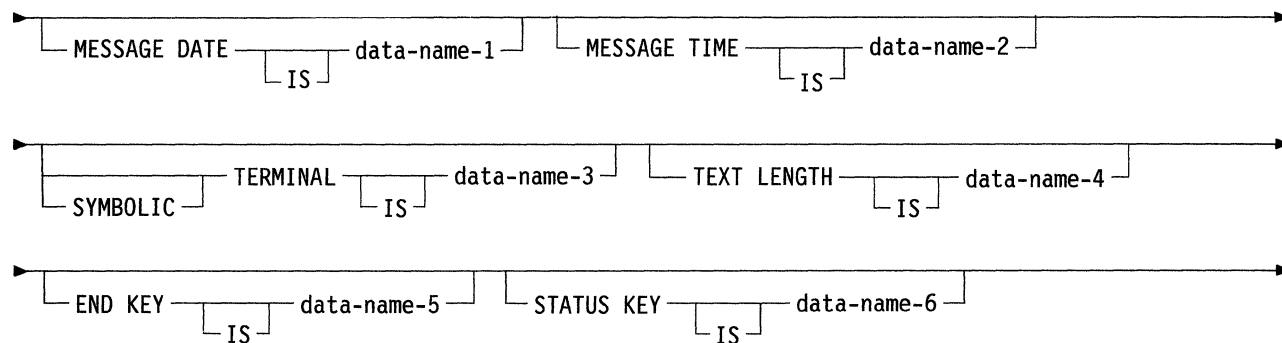
Format 2



Format 3



where phrase-1 is:



Syntax Rules

The following syntax rules apply to Data Division in the Communication Module:

All Formats

1. A communication description (CD) must appear only in the COMMUNICATION SECTION.

Formats 1 and 3

2. Within a single program, the INITIAL clause may be specified in only one CD. The INITIAL clause must not be used in a program that specifies the USING phrase of the Procedure Division Header. Refer to "Procedure Division Header" on page 11-33.

-
3. Except for the INITIAL clause, the optional clauses may be written in any order.
 4. If neither option in the format is specified, a level 01 data description entry must follow the CD description entry. Either option may be followed by a level 01 data description entry.
 5. Record description entries following an input CD implicitly redefine this record and must describe a record of exactly 87 standard data format characters. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. However, the MCS will always reference the record according to the data descriptions defined in syntax rule 4.
 6. data-name-1, data-name-2, ..., data-name-11 must be unique within the CD. Within this series, any data-name may be replaced by the reserved word FILLER.

Format 2

7. The optional clauses may be written in any order.
8. If none of the optional clauses of the CD is specified, a level 01 data description entry must follow the CD description entry.
9. Record descriptions following an output CD implicitly redefine this record. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. Note that the MCS will always reference the record according to the data descriptions defined in general rule 18 on page 15-12.
10. data-name-1, data-name-2, ..., data-name-5 must be unique within a CD.
11. If the DESTINATION TABLE OCCURS clause is not specified, one ERROR KEY and one SYMBOLIC DESTINATION area is assumed. In this case, neither subscripting nor indexing is permitted when referencing these data items.
12. If the DESTINATION TABLE OCCURS clause is specified, data-name-4 and data-name-5 may only be referred to by subscripting or indexing.
13. There is no restriction on the value of the data item referenced by data-name-1 and integer-2.

Format 3

14. Record descriptions following an input-output CD implicitly redefine this record and must describe a record of exactly 33 standard data characters. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. The MCS will always reference the record according to the data descriptions defined in rule 26 on page 15-13.
15. data-name-1, data-name-2, ..., data-name-6 must be unique within the CD. Within this series, any data-name may be replaced by the reserved word FILLER.

General Rules

The following general rules apply to Data Division in the Communication module:

All Formats

1. Figure 15-1 on page 15-8 indicates the possible contents of the data items referenced by data-name-10 for Format 1, data-name-3 for Format 2 and by data-name-6 for Format 3 at the completion of each statement shown. An X on a line in a statement column indicates that the associated code shown for that line is possible for that statement.

RECEIVE														
SEND input-output-cd														
SEND output-cd														
PURGE														
ACCEPT MESSAGE COUNT														
ENABLE INPUT														
ENABLE INPUT I/O TERMINAL														
ENABLE OUTPUT														
DISABLE INPUT														
DISABLE INPUT I/O TERMINAL														
DISABLE OUTPUT														
Status Key Value														
X	X	X	X	X	X	X	X	X	X	X	X	00	No error detected. Action completed.	
		X										10	One or more destinations are disabled. Action completed. (See General Rule 2.)	
	X	X										10	Destination disabled. No action taken.	
					X	X	X	X	X	X		15	Symbolic source, or one or more queues of destinations already disabled/enabled. (See General Rule 2.)	
	X	X	X				X				X	20	One or more destinations unknown. Action completed for known destination. (See General Rule 2.)	
X				X	X			X				20	One or more queues or subqueues unknown. No action taken.	
X						X			X			21	Symbolic source is unknown. No action taken.	
		X	X				X			X		30	Destination count invalid. No action taken.	
					X	X	X	X	X	X		40	Password invalid. No enabling/disabling action taken.	
	X	X										50	Text length exceeds size of identifier-1.	
	X	X										60	Portion requested to be sent has text length of zero or identifier-1 absent. No action taken.	
		X										65	Output queue capacity exceeded. See General Rule 2.	
			X									70	One or more destinations do not have portions associated with them. Action completed for other destinations.	
		X	X		X		X	X		X		80	A combination of at least two status key conditions 10, 15, and 20 have occurred. (See General Rule 2.)	
												9x	Operating-system-defined status.	

Figure 15-1. Communication Status Key Condition

2. Figure 15-2 indicates the possible content of the data item referenced by data-name-4 for Format 2 at the completion of each statement shown. An X on a line in a statement column indicates that the associated error key value shown for that line is possible for that statement.

SEND					
PURGE					
ENABLE OUTPUT					
DISABLE OUTPUT					
Error Key Value					
X	X	X	X	0	No error.
X	X	X	X	1	Symbolic destination unknown.
X	X			2	Symbolic destination disabled.
	X			4	No partial message with referenced symbolic destination.
		X	X	5	Symbolic destination already enabled/disabled.
X				6	Output queue capacity exceeded.
				7-9	Reserved for future use.
				A-Z	Operating-system-defined condition.

Figure 15-2. Error Key Values

Format 1

3. The input CD information constitutes the communication between the MCS and the program as information about the message being handled. This information does not come from the terminal as part of the message.
4. For each input CD, a record area of 87 contiguous standard data format characters is allocated. This record area is defined to the MCS as follows:
 - a. The SYMBOLIC QUEUE clause defines data-name-1 as the name of an elementary alphanumeric data item of 12 characters occupying positions 1-12 in the record.
 - b. The SYMBOLIC SUBQUEUE-1 clause defines data-name-2 as the name of an elementary alphanumeric data item of 12 characters occupying positions 13-24 in the record.
 - c. The SYMBOLIC SUBQUEUE-2 clause defines data-name-3 as the name of an elementary alphanumeric data item of 12 characters occupying positions 25-36 in the record.
 - d. The SYMBOLIC SUBQUEUE-3 clause defines data-name-4 as the name of an elementary alphanumeric data item of 12 characters occupying positions 37-48 in the record.
 - e. The MESSAGE DATE clause defines data-name-5 as the name of a data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 49-54 in the record.
 - f. The MESSAGE TIME clause defines data-name-6 as the name of a data item whose implicit description is that of an integer of 8 digits without an operational sign occupying character positions 55-62 in the record.
 - g. The SYMBOLIC SOURCE clause defines data-name-7 as the name of an elementary alphanumeric data item of 12 characters occupying positions 63-74 in the record.
 - h. The TEXT LENGTH clause defines data-name-8 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign occupying character positions 75-78 in the record.

- i. The END KEY clause defines data-name-9 as the name of an elementary alphanumeric data item of 1 character occupying position 79 in the record.
- j. The STATUS KEY clause defines data-name-10 as the name of an elementary alphanumeric data item of 2 characters occupying positions 80-81 in the record.
- k. The MESSAGE COUNT clause defines data-name-11 as the name of a data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 82 through 87 in the record.

The second option may be used to replace the above clauses by a series of data-names which, taken in order, correspond to the data-names defined by these clauses.

Use of either option results in a record whose implicit description is equivalent to the following:

IMPLICIT DESCRIPTION	COMMENT
01 data-name-0	
02 data-name-1 PICTURE X(12).	SYMBOLIC QUEUE
02 data-name-2 PICTURE X(12).	SYMBOLIC SUBQUEUE-1
02 data-name-3 PICTURE X(12).	SYMBOLIC SUBQUEUE-2
02 data-name-4 PICTURE X(12).	SYMBOLIC SUBQUEUE-3
02 data-name-5 PICTURE 9(06).	MESSAGE DATE
02 data-name-6 PICTURE 9(08).	MESSAGE TIME
02 data-name-7 PICTURE X(12).	SYMBOLIC SOURCE
02 data-name-8 PICTURE 9(04).	TEXT LENGTH
02 data-name-9 PICTURE X.	END KEY
02 data-name-10 PICTURE XX.	STATUS KEY
02 data-name-11 PICTURE 9(06).	MESSAGE COUNT

Note: The information in the column COMMENT is for clarification and is not part of the description.

- 5. The contents of the data items referenced by data-name-2, data-name-3, and data-name-4, when not being used must contain spaces.
- 6. The data items referenced by data-name-1, data-name-2, data-name-3, and data-name-4 contain symbolic names designating queues, subqueues, ..., respectively. All symbolic names must follow the rules for the formation of system-names, and must have been previously defined to the MCS.
- 7. A RECEIVE statement causes the serial return of the next message or a portion of a message from the queue as specified by the entries in the CD.

If during the execution of a RECEIVE statement, a message from a more specific source is needed, the contents of the data item referenced by data-name-1 can be made more specific by the use of the contents of the data items referenced by data-name-2, data-name-3, and in turn data-name-4. When a given level of the queue structure is specified, all higher levels must also be specified.

If fewer than all the levels of the queue hierarchy are specified, the MCS determines the *message* or portion of a message to be accessed.

After the execution of a RECEIVE statement, the contents of the data items referenced by data-name-1 through data-name-4 will contain the symbolic names of all the levels of the queue structure.

- 8. When a program is scheduled by the MCS to process a message, that program establishes a run-unit. The symbolic names of the queue structure that demanded the run-unit will be placed in the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause as applicable. In all other cases, the contents of the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause are initialized to spaces.

The symbolic names are inserted or the initialization to spaces is completed prior to the execution of the first Procedure Division statement.

The execution of a subsequent RECEIVE statement naming the same contents of the data items referenced by data-name-1 through data-name-4 will return the actual message that caused the program to be scheduled. Only at that time will the remainder of the CD be updated.

9. If the MCS attempts to schedule a program lacking an INITIAL clause, the results are undefined.
10. data-name-5 has the format YYMMDD (year, month, day). Its contents represent the date on which the MCS recognizes that the message is complete.

The contents of the data item referenced by data-name-5 are only updated by the MCS as part of the execution of a RECEIVE statement.

11. The contents of data-name-6 have the format HHMMSSSTT (hours, minutes, seconds, hundredths of a second) and its contents represent the time at which the MCS recognizes that the message is complete.

The contents of the data item referenced by data-name-6 are only updated by the MCS as part of the execution of the RECEIVE statement.

12. During the execution of a RECEIVE statement, the MCS provides, in the data item referenced by data-name-7, the symbolic name of the communications terminal that is the source of the message being transferred. This symbolic name must follow the rules for the formation of system names. However, if the symbolic name of the communication terminal is not known to the MCS, the contents of the data item referenced by data-name-7 will contain spaces.

13. The MCS indicates by the contents of the data item referenced by data-name-8 the number of character positions filled as a result of the execution of the RECEIVE statement. Refer to "RECEIVE Statement" on page 15-26.

14. The contents of the data item referenced by data-name-9 are set only by the MCS as part of the execution of a RECEIVE statement according to the following rules:

- a. When the RECEIVE MESSAGE phrase is specified, then data-name-9 is set to one of the following rules:

- If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3.
- If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2.
- If less than a message is transferred, the contents of the data item referenced by data-name-9 are set to 0.

- b. When the RECEIVE SEGMENT phrase is specified, data-name-9 is set to one of the following:

- If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3.
- If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2.
- If an end of segment has been detected, the contents of the data item referenced by data-name-9 are set to 1.
- If less than a message segment is transferred, the contents of the data item referenced by data-name-9 are set to 0.

- c. When more than one of the above conditions is satisfied simultaneously, the rule first satisfied in the order listed determines the contents of the data item referenced by data-name-9.

15. The contents of the data item referenced by data-name-10 indicate the status condition of the previously executed RECEIVE, ACCEPT MESSAGE COUNT, ENABLE INPUT, or DISABLE INPUT statements.

The actual association between the contents of the data item referenced by data-name-10 and the status condition itself is defined in Figure 15-1 on page 15-8.

16. The contents of the data item referenced by data-name-11 indicate the number of messages that exist in a queue, subqueue-1, ... The MCS updates the contents of the data item referenced by data-name-11 only as part of the execution of an ACCEPT statement with the COUNT phrase.

Format 2

17. The output CD information is not sent to the terminal, but constitutes the communication between the program and the MCS as information about the message being handled.
18. For each output CD, a record area of contiguous standard data format characters is allocated according to the formula (10 plus 13 times integer-2).
 - a. The DESTINATION COUNT clause defines data-name-1 as the name of a data item whose implicit description is that of an integer without an operational sign occupying character positions 1-4 in the record.
 - b. The TEXT LENGTH clause defines data-name-2 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign occupying character positions 4-8 in the record.
 - c. The STATUS KEY clause defines data-name-3 to be an elementary alphanumeric data item of 2 characters occupying positions 9-10 in the record.
 - d. Character positions 11-23 and every set of 13 characters thereafter will form table items of the following description:
 - The ERROR KEY clause defines data-name-4 as the name of an elementary alphanumeric data item of 1 character.
 - The SYMBOLIC DESTINATION clause defines data-name-5 as the name of an elementary alphanumeric data item of 12 characters.

Use of the above clauses results in a record whose implicit description is equivalent to:

IMPLICIT DESCRIPTION	COMMENT
01 data-name-0	
02 data-name-1 PICTURE 9(04).	DESTINATION COUNT
02 data-name-2 PICTURE 9(04).	TEXT LENGTH
02 data-name-3 PICTURE XX.	STATUS KEY
02 data-name OCCURS integer-2 TIMES.	DESTINATION TABLE
03 data-name-4 PICTURE X.	ERROR KEY
03 data-name-5 PICTURE X(12).	SYMBOLIC DESTINATION

Note: In the above, the information under COMMENT is for clarification and is not part of the description.

19. During the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement, the contents of the data item referenced by data-name-1 will indicate to the MCS the number of symbolic destinations that are to be used from the area referenced by data-name-5.

The MCS finds the first symbolic destination in the first occurrence of the area referenced by data-name-5, the second symbolic destination in the second occurrence of the area referenced by data-name-5 ..., up to and including the occurrence of the area referenced by data-name-5 and indicated by the contents of data-name-1.

If during the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement the value of the data item referenced by data-name-1 is outside the range of 1 through integer-2, an error condition is indicated and the execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement is terminated.

-
20. The user is responsible for ensuring that the value of the data item referenced by data-name-1 is valid at the time of execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.
 21. As part of the execution of a SEND statement, the MCS will interpret the contents of the data item referenced by data-name-2 to be the user's indication of the number of leftmost character positions of the data item referenced by the associated SEND identifier from which data is to be transferred. Refer to "SEND Statement" on page 15-29.
 22. Each occurrence of the data item referenced by data-name-5 contains a symbolic destination previously known to the MCS. These symbolic destination names must follow the rules for the formation of system-names.
 23. The contents of the data item referenced by data-name-3 indicate the status condition of the previously executed SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.
The actual association between the contents of the data item referenced by data-name-3 and the status condition itself is defined in Figure 15-1 on page 15-8.

24. If, during the execution of a SEND, an ENABLE OUTPUT, or a DISABLE OUTPUT statement, the MCS determines that any specified destination is unknown, the contents of the data item referenced by data-name-3, and all occurrences of the data items referenced by data-name-4, are updated.
The contents of the data item referenced by data-name-4 when equal to 1 indicate that the associated value in the area referenced by data-name-5 has not been previously defined to the MCS. Otherwise, the contents of the data item referenced by data-name-4 are set to zero.

Format 3

25. The input-output CD information constitutes the communication between the MCS and the program about the message being handled. This information does not come from the terminal as part of the message.
26. For each input-output CD, a record area of 33 contiguous character positions is allocated. This record area is defined to the MCS as follows:
 - a. The MESSAGE DATE clause defines data-name-1 as the name of a data item whose implicit description is that of an integer of 6 digits, without an operational sign, occupying positions 1-6 in the record.
 - b. The MESSAGE TIME clause defines data-name-2 as the name of a data item whose implicit description is that of an integer of 8 digits, without an operational sign, occupying positions 7-14 in the record.
 - c. The SYMBOLIC TERMINAL clause defines data-name-3 as the name of an elementary alphanumeric data item of 12 characters occupying positions 15-26 in the record.
 - d. The TEXT LENGTH clause defines data-name-4 as the name of an elementary data item whose implicit description is that of an integer of 4 digits, without an operational sign, occupying positions 27-30 in the record.
 - e. The END KEY clause defines data-name-5 as the name of an elementary alphanumeric data item of 1 character occupying position 31 in the record.
 - f. The STATUS KEY clause defines data-name-6 as the name of an elementary alphanumeric data item of 2 characters occupying positions 32 and 33 in the record.

The second option may be used to replace the above clauses by a series of data names which, taken in order, correspond to the data names defined by these clauses.

Use of either option results in a record whose implicit description is equivalent to the following:

IMPLICIT DESCRIPTION	COMMENT
01 data-name-0	
02 data-name-1 PICTURE 9(6).	MESSAGE DATE
02 data-name-2 PICTURE 9(8).	MESSAGE TIME
02 data-name-3 PICTURE X(12).	SYMBOLIC TERMINAL
02 data-name-4 PICTURE 9(4).	TEXT LENGTH
02 data-name-5 PICTURE X.	END KEY
02 data-name-6 PICTURE XX.	STATUS KEY

Note: In the above, the information under COMMENT is for clarification and is not part of the data description.

27. When a program is scheduled by the MCS to process a message, the first RECEIVE statement referencing the input-output CD with the INITIAL clause returns the actual message that caused the program to be scheduled.

28. data-name-1 has the format YYMMDD (year, month, day). Its content represents the date on which the MCS recognizes that the message is complete.

The content of the data item referenced by data-name-1 is updated only by the MCS as part of the execution of a RECEIVE statement.

29. data-name-2 has the format HHMMSSSTT (hours, minutes, seconds, hundredths of a second) and its content represents the time at which the MCS recognizes that the message is complete.

The content of the data item referenced by data-name-2 is updated only by the MCS as part of the execution of the RECEIVE statement.

30. When a program is scheduled by the MCS to process a message, that program establishes a run-unit. The symbolic name of the communication terminal that is the source of the message that invoked this program is placed in the data item referenced by data-name-3 of the input-output CD associated with the INITIAL clause as applicable. This symbolic name must follow the rules for the formation of system-names.

In all other cases, the content of the data item referenced by data-name-3 of the input-output CD associated with the INITIAL clause is initialized to spaces.

The symbolic name is inserted, or the initialization to spaces is completed, prior to the execution of the first Procedure Division statement.

31. If the MCS attempts to schedule a program lacking an INITIAL clause, the results are undefined.

32. When the INITIAL clause is specified for an input-output CD and the program is scheduled by the MCS, the content of the data item referenced by data-name-3 must not be changed by the program. If this content is changed, the execution of any statement referencing cd-name-1 is unsuccessful and the data item referenced by data-name-6 is set to indicate unknown source or destination, as applicable. Refer to rule 11 on page 15-11.

33. For an input-output CD without the INITIAL clause, or for an input-output CD with the INITIAL clause when the program is not scheduled by the MCS, the program must specify the symbolic name of the source or destination in data-name-3 prior to the execution of the first statement referencing cd-name-1.

After executing the first statement referencing cd-name-1, the content of the data item referenced by data-name-3 must not be changed by the program. If this content is changed, the execution of any statement referencing cd-name-1 is unsuccessful and the data item referenced by data-name-6 is set to indicate unknown source or destination, as applicable. Refer to general rule 11 on page 15-11.

-
34. The MCS indicates, through the content of the data item referenced by data-name-4, the number of character positions filled as a result of the RECEIVE statement. Refer to "RECEIVE Statement" on page 15-26.

As part of the execution of a SEND statement, the MCS interprets the content of the data item referenced by data-name-4 as an indication of the number of leftmost character positions of the data item referenced by the associated SEND identifier from which data is transferred. Refer to "SEND Statement" on page 15-29.

35. The content of the data item referenced by data-name-5 is set only by the MCS as part of the execution of a RECEIVE statement according to the following rules:

a. When the RECEIVE MESSAGE phrase is specified:

- If an end of group has been detected, the content of the data item referenced by data-name-5 is set to 3.
- If an end of message has been detected, the content of the data item referenced by data-name-5 is set to 2.
- If less than a message segment is transferred, the content of the data item referenced by data-name-5 is set to 0.

b. When the RECEIVE SEGMENT phrase is specified:

- If an end of group has been detected, the content of the data item referenced by data-name-5 is set to 3.
- If an end of message has been detected, the content of the data item referenced by data-name-5 is set to 2.
- If less than a message segment is transferred, the content of the data item referenced by data-name-5 is set to 0.

c. When more than one of the conditions is satisfied simultaneously, the first rule satisfied in the order listed determines the content of the data item referenced by data-name-5.

36. The content of the data item referenced by data-name-6 indicates the status condition of the previously executed DISABLE, ENABLE, PURGE, RECEIVE, or SEND statement.

The actual association between the content of the data item referenced by data-name-6 and the status condition itself is defined in general rule 11 on page 15-11.

Example 1

The following is an example of a Communication Description followed by Record Description entries. The length and format of each field in this example is fixed and the total length must be exactly 87 standard data format characters.

COMMUNICATION SECTION.

```
CD IN-QUE
    FOR INITIAL INPUT.
01 IN-QUE-AREA.
02 IN-QUES.
    04 IN-Q      PIC X(12).
    04 IN-SUBQ1  PIC X(12).
    04 IN-SUBQ2  PIC X(12).
    04 IN-SUBQ3  PIC X(12).
02 IN-MSG-DATE  PIC 9(06).
02 IN-MSG-TIME  PIC 9(08).
02 IN-SYM-SOURCE PIC X(12).
02 IN-TXT-LENGTH PIC 9(04).
02 IN-END-KEY   PIC X.
02 IN-STATUS-KEY PIC XX.
02 IN-MSG-COUNT PIC 9(06).
```

Example 2

The CD in this example is semantically equivalent to Example 1. Key words are spelled out. Length and format of fields are implied.

COMMUNICATION SECTION.

```
CD IN-QUE
    FOR INITIAL INPUT
    SYMBOLIC QUEUE IS IN-Q
    SYMBOLIC SUB-QUEUE-1 IS IN-SUBQ1
    SYMBOLIC SUB-QUEUE-2 IS IN-SUBQ2
    SYMBOLIC SUB-QUEUE-3 IS IN-SUBQ3
    MESSAGE DATE IS IN-MSG-DATE
    MESSAGE TIME IS IN-MSG-TIME
    SYMBOLIC SOURCE IS IN-SYM-SOURCE
    TEXT LENGTH IS IN-TXT-LENGTH
    END KEY IS IN-END-KEY
    STATUS KEY IS IN-STATUS-KEY
    MESSAGE COUNT IS IN-MSG-COUNT.
```

Example 3

The following CD is semantically equivalent to Examples 1 and 2. Eleven data names are presented with no leading key words as a short form of Example 2.

COMMUNICATION SECTION.

```
CD IN-QUE
  FOR INITIAL INPUT
    IN-Q IN-SUBQ1 IN-SUBQ2 IN-SUBQ3 IN-MSG-DATE IN-MSG-TIME
    IN-SYM-SOURCE IN-TXT-LENGTH IN-END-KEY IN-STATUS-KEY
    IN-MSG-COUNT.
```

Example 4

Examples 4 and 5 describe two output communication descriptions. The format of Example 4 allows initial values to be associated with data names.

```
CD OUT-QUE FOR OUTPUT.
01 OUT-QUE-AREA.
  02 OUT-DEST-COUNT PIC 9(04) VALUE 1.
  02 OUT-TXT-LENGTH PIC 9(04).
  02 OUT-STATUS-KEY PIC XX.
  02 OUT-DEST-ERKEY PIC X.
  02 OUT-SYM-DEST PIC X(12) VALUE "CONSOL1".
```

Example 5

```
CD OUT-QUE FOR OUTPUT
  DESTINATION COUNT IS OUT-DEST-COUNT
  TEXT LENGTH IS OUT-TXT-LENGTH
  STATUS KEY IS OUT-STATUS-KEY
  DESTINATION TABLE OCCURS 1 TIMES
  ERROR KEY IS OUT-DEST-ERKEY
  SYMBOLIC DESTINATION IS OUT-SYM-DEST.
```

Example 6

Examples 6 through 8 represent three equivalent I-O communication description formats.

```
CD IO-QUE FOR I-O
  MESSAGE DATE IS IO-MSG-DATE
  MESSAGE TIME IS IO-MSG-TIME
  SYMBOLIC TERMINAL IS IO-SYM-TERM
  TEXT LENGTH IS IO-TXT-LENGTH
  END KEY IS IO-END-KEY
  STATUS KEY IS IO-STATUS-KEY.
```

Example 7

```
CD  IO-QUE FOR I-0
    IO-MSG-DATE IO-MSG-TIME IO-SYM-TERM IO-TXT-LENGTH
    IO-END-KEY IO-STATUS-KEY.
```

Example 8

```
CD  IO-QUE FOR I-0
01  IO-QUE-AREA.
    02 IO-MSG-DATE PIC 9(6).
    02 IO-MSG-TIME PIC 9(8).
    02 IO-SYM-TERM PIC X(12).
    02 IO-TXT-LENGTH PIC 9(4).
    02 IO-END-KEY PIC X.
    02 IO-STATUS-KEY PIC XX.
```

Procedure Division in the Communication Module

Six Procedure Division statements are used to perform communication tasks. These statements, as listed below, provide the capabilities to enable/disable communication, to send/receive messages, to get a message count, and to purge a message queue.

- ACCEPT MESSAGE COUNT
- DISABLE
- ENABLE
- PURGE
- RECEIVE
- SEND.

ACCEPT MESSAGE COUNT Statement

Function

The ACCEPT MESSAGE COUNT statement causes the number of messages in a queue to be made available.

General Format

The following figure shows the format of the ACCEPT MESSAGE COUNT statement:

▶— ACCEPT cd-name MESSAGE COUNT —▶

Syntax Rule

cd-name must reference an input CD.

General Rules

1. The ACCEPT MESSAGE COUNT statement causes the MESSAGE COUNT field specified for cd-name to be updated to indicate the number of messages that exist in a queue, subqueue-1,
2. Upon execution of the ACCEPT MESSAGE COUNT statement, the contents of the area specified by a communication description entry must contain at least the name of the symbolic queue to be tested. Testing the condition causes the contents of the data items referenced by data-name-10 (STATUS KEY) and data-name-1 (MESSAGE COUNT) of the area associated with the communication entry to be appropriately updated. Refer to “Communication Description — Complete Entry Skeleton” on page 15-4.

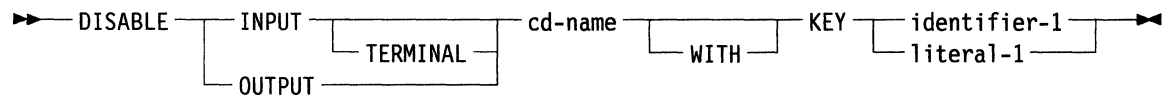
DISABLE Statement

Function

The **DISABLE** statement notifies the MCS to inhibit data transfer between specified output queues and destinations or between specified sources and input queues.

General Format

The following figure shows the format of the **DISABLE** statement:



Syntax Rules

The following syntax rules apply to the **DISABLE** statement:

1. **cd-name** must reference an input CD when the **INPUT** phrase is specified.
2. **cd-name** must reference an output CD when the **OUTPUT** phrase is specified.
3. **literal-1** or the contents of the data item referenced by **identifier-1** must be defined as alphanumeric.

General Rules

The following general rules apply to the **DISABLE** statement:

1. The **DISABLE** statement provides a logical disconnection between the MCS and the specified sources or destinations. When this logical disconnection is already in existence, or is to be handled by some other means external to this program, the **DISABLE** statement is not required in this program. The logical path for the transfer of data between the COBOL programs and the MCS is not affected by the **DISABLE** statement.
2. When the **INPUT** phrase with the optional word **TERMINAL** is specified, the logical path between the source and all queues and subqueues is deactivated. Only the contents of the data item referenced by **data-name-7** (**SYMBOLIC SOURCE**) of the area referenced by **cd-name** are meaningful.
3. When the **INPUT** phrase without the optional word **TERMINAL** is specified, the logical paths for all of the sources associated with the queues and subqueues specified by the contents of **data-name-1** (**SYMBOLIC QUEUE**) through **data-name-4** (**SYMBOLIC SUBQUEUE-3**) of the area referenced by **cd-name** are deactivated.
4. When the **OUTPUT** phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by **data-name-5** (**SYMBOLIC DESTINATION**) of the area referenced by **cd-name** are deactivated.

-
5. literal-1 or the contents of the data-name referenced by identifier-1 will be matched with a password built into the system. The DISABLE statement will be honored only if literal-1 or the contents of the data item referenced by identifier-1 match the system password. When literal-1 or the contents of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name is updated.

The MCS must be capable of handling a password of from 1 to 10 characters inclusive.

6. The MCS will ensure that the execution of a DISABLE statement will cause the logical disconnection at the earliest time the source or destination is inactive. The execution of the DISABLE statement will never cause the remaining portion of the message to be terminated during transmission to or from a terminal.

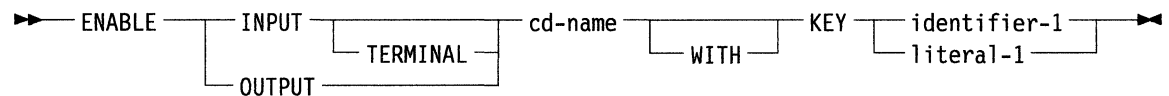
ENABLE Statement

Function

The **ENABLE** statement notifies the MCS to allow data transfer between specified output queues and destinations for output or between specified sources and input queues for input.

General Format

The following figure shows the format of the **ENABLE** statement:



Syntax Rules

The following syntax rules apply to the **ENABLE** statement:

1. **cd-name** must reference an input CD when the **INPUT** phrase is specified.
2. **cd-name** must reference an output CD when the **OUTPUT** phrase is specified.
3. **literal-1** or the contents of the data item referenced by **identifier-1** must be defined as alphanumeric.

General Rules

The following general rules apply to the **ENABLE** statement:

1. The **ENABLE** statement provides a logical connection between the MCS and the specified sources or destinations. When this logical connection is already in existence, or is to be handled by some other means external to this program, the **ENABLE** statement is not required in this program. The logical path for the transfer of data between the COBOL programs and the MCS is not affected by the **ENABLE** statement.
2. When the **INPUT** phrase with the optional word **TERMINAL** is specified, the logical path between the source and all associated queues and subqueues which are already enabled is activated. Only the contents of the data item referenced by **data-name-7** (**SYMBOLIC SOURCE**) of the area referenced by **cd-name** are meaningful to the MCS.
3. When the **INPUT** phrase without the optional word **TERMINAL** is specified, the logical paths for all of the sources associated with the queue and subqueues specified by the contents of **data-name-1** (**SYMBOLIC QUEUE**) through **data-name-4** (**SYMBOLIC SUBQUEUE-3**) of the area referenced by **cd-name** are activated.
4. When the **OUTPUT** phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by **data-name-5** (**SYMBOLIC DESTINATION**) of the area referenced by **cd-name** are activated.

-
5. literal-1 or the contents of the data item referenced by identifier-1 will be matched with a password built into the system. The `ENABLE` statement will be honored only if literal-1 or the contents of the data item referenced by identifier-1 match the system password. When literal-1 or the contents of the data item referenced by identifier-1 do not match the system password, the value of the `STATUS KEY` item in the area referenced by `cd-name` is updated.

The MCS must be capable of handling a password of from 1 to 10 characters inclusive.

PURGE Statement

Function

The PURGE statement eliminates from the MCS a partial message that has been released by one or more SEND statements.

General Format

The following figure shows the format of the PURGE statement:

►► — PURGE cd-name — ◄◄

Syntax Rule

cd-name must reference an output CD or input-output CD.

General Rules

The following general rules apply to the PURGE statement:

1. Execution of a PURGE statement causes the MCS to eliminate any partial message awaiting transmission to the destinations specified in the CD referenced by cd-name.
2. Any message that has associated with it an EMI or EGI is not affected by the execution of a PURGE statement.
3. The content of the status key data item and the content of the error key data item (if applicable) of the area referenced by cd-name-1 are updated by the MCS. Refer to “Communication Description — Complete Entry Skeleton” on page 15-4.

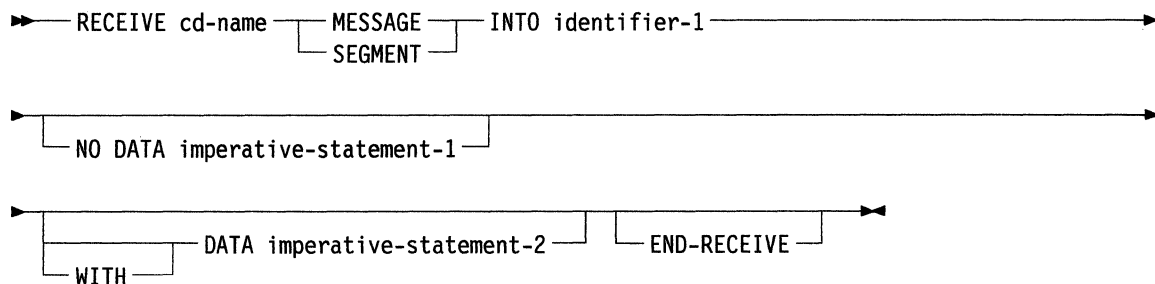
RECEIVE Statement

Function

The **RECEIVE** statement makes available to the COBOL program a message, message segment, or a portion of a message or segment, along with pertinent information about that data from a queue maintained by the Message Control System. The **RECEIVE** statement allows for a specific imperative statement when no data is available.

General Format

The following figure shows the format for the **RECEIVE** statement:



Syntax Rule

cd-name must reference an input CD or input-output CD.

General Rules

The following general rules apply to the **RECEIVE** statement:

1. If cd-name references an input CD, the contents of the data items specified by data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUBQUEUE-3) of the area referenced by cd-name designate the queue structure containing the message. Refer to “Communication Description – Complete Entry Skeleton” on page 15-4.
2. If cd-name references an input-output CD, the contents of the data items specified by data-name-3 (SYMBOLIC TERMINAL) of the area referenced by cd-name designate the source of the message. Refer to “Communication Description – Complete Entry Skeleton” on page 15-4.
3. The message, message segment, or portion of a message or segment is transferred to the receiving character positions of the area referenced by identifier-1 and aligned to the left without space fill.

-
4. When, during the execution of a RECEIVE statement, the MCS makes data available in the data item referenced by identifier-1, the NO DATA phrase, if specified, is ignored and control is transferred to the end of the RECEIVE statement or, if the WITH DATA phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the RECEIVE statement.
 5. When, during the execution of a RECEIVE statement, the MCS does not make data available in the data item referenced by identifier-1, one of the three actions listed below will occur. The conditions under which data is not made available are defined by the implementer.
 - a. If the NO DATA phrase is specified in the RECEIVE statement, the RECEIVE operation is terminated with the indication that action is complete and 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 RECEIVE statement and the WITH DATA phrase, if specified, is ignored.
 - b. If the NO DATA phrase is not specified in the RECEIVE statement, execution of the object program is suspended until data is made available in the data item referenced by identifier-1.
 - c. If one or more queues or subqueues are unknown to the MCS, the appropriate status key code is stored and control is then transferred as if data had been made available.
 6. The data items identified by cd-name are appropriately updated by the MCS at each execution of a RECEIVE statement.
 7. A single execution of a RECEIVE statement never returns to the data item referenced by identifier-1 more than a single message (when the MESSAGE phrase is used) or a single segment (when the SEGMENT phrase is used). However, the MCS does not pass any portion of a message to the object program until the entire message is available in the input queue, even if the SEGMENT phrase of the RECEIVE statement is specified.
 8. When the MESSAGE phrase is used, end of segment indicators are ignored, and the following rules apply to the data transfer:
 - a. If a message is the same size as the area referenced by identifier-1, the message is stored in the area referenced by identifier-1.
 - b. If a message size is less than the area referenced by identifier-1, the message is aligned to the leftmost character position of the area referenced by identifier-1 with no space fill.
 - c. If a message size is greater than the area referenced by identifier-1, the message fills the area referenced by identifier-1 left to right starting with the leftmost character of the message. The remainder of the message can be transferred to the area referenced by identifier-1 with subsequent RECEIVE statements referring to the same queue, subqueue, The remainder of the message, for the purposes of applying rules 8a, 8b, and 8c is treated as a new message.
 9. When the SEGMENT phrase is used, the following rules apply:
 - a. If a segment is the same size as the area referenced by identifier-1, the segment is stored in the area referenced by identifier-1.
 - b. If a segment size is less than the area referenced by identifier-1, the segment is aligned to the leftmost character position of the area referenced by identifier-1 with no space fill.

-
- c. If a segment size is greater than the area referenced by identifier-1, the segment fills the area referenced by identifier-1 left to right starting with the leftmost character of the segment. The remainder of the segment can be transferred to the area referenced by identifier-1 with subsequent RECEIVE statements referring to the same queue, subqueue, The remainder of the segment, for the purposes of applying rules 9a on page 15-27, 9b on page 15-27, and 9c is treated as a new segment.
 - d. If the text to be accessed by the RECEIVE statement has associated with it an end of message indicator or end of group indicator, the existence of an end of segment indicator associated with the text is implied and the text is treated as a message segment.
10. Once the execution of a RECEIVE statement has returned a portion of a message, only subsequent execution of RECEIVE statements in that run-unit can cause the remaining portion of the message to be returned.
 11. The END-RECEIVE phrase delimits the scope of the RECEIVE statement. Refer to "Delimited Scope Statements" on page 2-48.

SEND Statement

Function

The SEND statement causes a message, a message segment, or a portion of a message or segment to be released to one or more output queues maintained by the Message Control System.

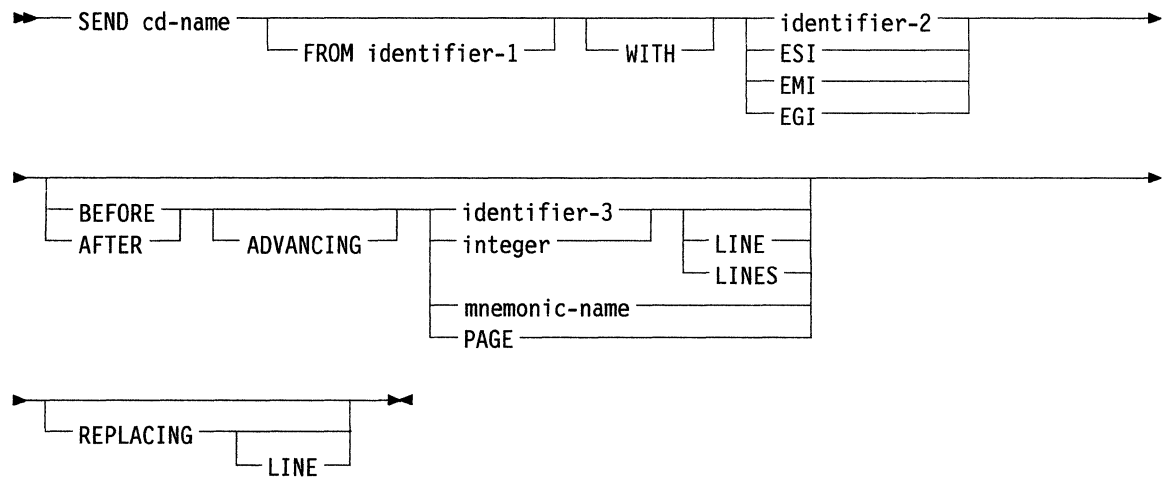
General Format

The following figures show the format of the SEND statement:

Format 1

► SEND cd-name FROM identifier-1 ◄

Format 2



Syntax Rules

The following syntax rules apply to the SEND statement:

1. cd-name must reference an output CD or an input-output CD.
2. identifier-2 must reference a one-character integer without an operational sign.
3. When identifier-3 is used in the ADVANCING phrase, it must be the name of an elementary integer item.
4. When the mnemonic-name phrase is used, the name is identified with a particular feature specified. The mnemonic-name is defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.
5. An integer or the value of the data item referenced by identifier-3 may be zero.

General Rules

The following general rules apply to the SEND statement:

Both Formats

1. When a receiving communication device (printer, display screen, card punch, etc.) is oriented to a fixed line size:
 - a. Each message or message segment will begin at the leftmost character position of the physical line.
 - b. A message or message segment that is smaller than the physical line size is released and appears space-filled to the right.
 - c. Excess characters of a message or message segment will not be truncated. Characters will be packed to a size equal to that of the physical line and then transmitted to the device. The process continues on the next line with the excess characters.
2. When a receiving communication device (paper tape punch, another computer, etc.) is oriented to handle variable-length messages, each message or message segment will begin on the next available character position of the communications device.
3. As part of the execution of a SEND statement, the MCS will interpret the contents of the data item, referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name, to the user's specification of the number of leftmost character positions of the data item referenced by identifier-1, from which data is to be transferred.

If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are zero, no characters of the data item referenced by identifier-1 are transferred.

If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are outside the range of zero through the size of the data item referenced by identifier-1 inclusive, an error is indicated by the value of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred. Refer to Figure 15-1 on page 15-8 for Status.

4. As part of the execution of a SEND statement, the contents of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name is updated by the MCS. Refer to "Communication Description — Complete Entry Skeleton" on page 15-4.
5. The effect of having special control characters within the contents of the data item referenced by identifier-1 is undefined.
6. A single execution of a SEND statement for Format 1 releases only a single portion of a message or of a message segment to the MCS.

A single execution of a SEND statement of Format 2 never releases to the MCS more than a single message or a single message segment as indicated by the contents of the data item referenced by identifier-2 or by the specified indicator ESI, EMI, or EGI.

However, the MCS will not transmit any portion of a message to a communications device until the entire message is placed in the output queue.

7. During the execution of the run-unit, the disposition of a portion of a message not terminated by an EMI or EGI or which has not been eliminated by the execution of a PURGE statement is undefined. However, the message does not logically exist for the MCS and cannot be sent to a destination.
8. Once the execution of a SEND statement has released a portion of a message to the MCS, only subsequent execution of SEND statements in the same run-unit can cause the remaining portion of the message to be released.

Format 2

9. The contents of the data item referenced by identifier-2 indicate that the contents of the data item referenced by identifier-1 are to have associated with it an end of segment indicator, an end of message indicator, or an end of transmission indicator according to the following schedule:

If the contents of the data item referenced by identifier-2 associated with it is:	The contents of data item referenced by identifier-1 have:	Which means:
0	no indicator	no indicator
1	ESI	an end of segment indicator
2	EMI	an end of message indicator
3	EGI	an end of group indicator
Any character other than 1, 2, or 3 will be interpreted as 0		
If the contents of the data item referenced by identifier-2 is other than 1, 2, or 3, and identifier-1 is not specified, then an error is indicated by the value in the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred.		

10. The ESI indicates to the MCS that the message segment is complete. The EMI indicates to the MCS that the message is complete.
- The EGI indicates to the MCS that the group of messages is complete. The Run Time Environment specifies the interpretation that is given to the EGI by the MCS.
- The MCS will recognize these indications and establish whatever is necessary to maintain group, message, and segment control.
11. The hierarchy of ending indicators is EGI, EMI, ESI. An EGI need not be preceded by an ESI or EMI. An EMI need not be preceded by an ESI.
12. The ADVANCING phrase allows control of the vertical positioning of each message or message segment on a communication device where vertical positioning is applicable. If vertical positioning is not applicable on the device, the MCS will ignore the vertical positioning specified or implied.
13. If identifier-2 is specified and the content of the data item referenced by identifier-2 is zero, the ADVANCING phrase and the REPLACING phrase, if specified, are ignored by the MCS.
14. On a device where vertical positioning is applicable and the ADVANCING phrase is not specified, automatic advancing is provided as if AFTER ADVANCING 1 LINE had been specified.

-
15. If the **ADVANCING** phrase is implicitly or explicitly specified and vertical positioning is applicable, the following rules apply:
 - a. If identifier-3 or integer is specified, characters transmitted to the communication device will be repositioned vertically downward the number of lines equal to the value associated with the data item referenced by identifier-3 or integer.
 - b. If the value of the data item referenced by identifier-3 is negative, the results are undefined.
 - c. If mnemonic-name is specified, characters transmitted to the communication device will be positioned according to the rules specified for that communication device.
 - d. If the **BEFORE** phrase is used, the message or message segment is represented on the communication device before vertical repositioning according to 15a and 15b.
 - e. If the **AFTER** phrase is used, the message or message segment is represented on the communication device after vertical repositioning according to 15a and 15b.
 - f. If **PAGE** is specified, characters transmitted to the communication device will be represented on the device before or after (depending upon the phrase used) the device is repositioned to the next page. If **PAGE** is specified, then advancing is provided as if the user had specified **BEFORE** or **AFTER** (depending upon the phrase used) **ADVANCING 1 LINE** (**PAGE** has no meaning in conjunction with a specific device).
 16. When using receiving character-imaging communication devices, on which it is possible to present two or more characters at the same position, superimpose second or subsequent characters on characters already displayed at that position, or display each character in the place of characters previously transmitted to that line, the following rules apply:
 - a. If the **REPLACING** phrase is specified, the characters transmitted by the **SEND** statement replace all characters which may have previously been transmitted to the same line, beginning with the leftmost character position of the line.
 - b. If the **REPLACING** phrase is not specified, the characters transmitted by the **SEND** statement appear superimposed upon the characters that may have previously been transmitted to the same line, beginning with the leftmost character position of the line.
 17. When a receiving communication device does not support the replacement of characters, regardless of whether or not the **REPLACING** phrase is specified, the characters transmitted by the **SEND** statement appear superimposed upon the characters which may have previously been transmitted to the same line, beginning with the leftmost character position of the line.
 18. When a receiving communication device does not support the superimposing of two or more characters at the same position, regardless of whether or not the **REPLACING** phrase is specified, the characters transmitted by the **SEND** statement replace all characters that may have previously been transmitted to the same line beginning with the leftmost character position of the line.

Communication Sample Program

The following example shows a sample Communication program.

IDENTIFICATION DIVISION.
PROGRAM-ID. TESTCOMM.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-RISC-6000.
OBJECT-COMPUTER. IBM-RISC-6000.

DATA DIVISION.

WORKING-STORAGE SECTION.

```
01 NAK-MSG1          PIC X(20) VALUE "NO DATA RECEIVED".
01 ACK-MSG2          PIC X(22) VALUE "MESSAGE ACKNOWLEDGED".

01 ACK-MSG3.
   02 FILLER          PIC X(05) VALUE "QUE: ".
   02 ACK-QUES        PIC X(48).
   02 FILLER          PIC X(05) VALUE "DEV: ".
   02 ACK-DEVICE      PIC X(12).

01 MSG-HOLD          PIC X(72).
01 NAK-MSG1-LENGTH  PIC 99   VALUE 20.
01 ACK-MSG2-LENGTH  PIC 99   VALUE 22.
01 ACK-MSG3-LENGTH  PIC 99   VALUE 70.
```

COMMUNICATION SECTION.

```
CD IN-QUE
   FOR INITIAL INPUT.
01 IN-QUE-AREA.
   02 IN-QUES.
      04 IN-Q          PIC X(12).
      04 IN-SUBQ1      PIC X(12).
      04 IN-SUBQ2      PIC X(12).
      04 IN-SUBQ3      PIC X(12).
   02 IN-MSG-DATE      PIC 9(06).
   02 IN-MSG-TIME      PIC 9(08).
   02 IN-SYM-SOURCE    PIC X(12).
   02 IN-TXT-LENGTH    PIC 9(04).
   02 IN-END-KEY       PIC X.
   02 IN-STATUS-KEY    PIC XX.
   02 IN-MSG-COUNT     PIC 9(06).

CD OUT-QUE FOR OUTPUT.
01 OUT-QUE-AREA.
   02 OUT-DEST-COUNT   PIC 9(04)   VALUE 1.
   02 OUT-TXT-LENGTH   PIC 9(04).
   02 OUT-STATUS-KEY   PIC XX.
   02 OUT-DEST-ERKEY   PIC X.
   02 OUT-SYM-DEST     PIC X(12)   VALUE "CONSOL1".
```

PROCEDURE DIVISION.

ENABLE-MESSAGE.

ENABLE OUTPUT OUT-QUE WITH KEY "COMMTEST1".
ACCEPT IN-QUE MESSAGE COUNT.
PERFORM PROCESS-MESSAGES UNTIL IN-MSG-COUNT = 0.
STOP RUN.

PROCESS-MESSAGES.

RECEIVE IN-QUE MESSAGE INTO MSG-HOLD
NO DATA
MOVE NAK-MSG1-LENGTH TO OUT-TXT-LENGTH
SEND OUT-QUE FROM NAK-MSG1 WITH EMI
STOP RUN
END-RECEIVE.

MOVE ACK-MSG2-LENGTH TO OUT-TXT-LENGTH.
SEND OUT-QUE FROM ACK-MSG2 WITH EMI.

MOVE IN-QUES TO ACK-QUES.
MOVE IN-SYM-SOURCE TO ACK-DEVICE.
MOVE ACK-MSG3-LENGTH TO OUT-TXT-LENGTH.
SEND OUT-QUE FROM ACK-MSG3 WITH EMI.

MOVE IN-TXT-LENGTH TO OUT-TXT-LENGTH.
SEND OUT-QUE FROM MSG-HOLD WITH EGI.

ACCEPT IN-QUE MESSAGE COUNT.

IDENTIFICATION DIVISION.

PROGRAM-ID. TESTCOMM.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-RISC-6000.

OBJECT-COMPUTER. IBM-RISC-6000.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 NAK-MSG1 PIC X(20) VALUE "NO DATA RECEIVED".
01 ACK-MSG2 PIC X(22) VALUE "MESSAGE ACKNOWLEDGED".

01 ACK-MSG3.
02 FILLER PIC X(05) VALUE "QUE: ".
02 ACK-QUES PIC X(48).
02 FILLER PIC X(05) VALUE "DEV: ".
02 ACK-DEVICE PIC X(12).

01 MSG-HOLD PIC X(72).
01 NAK-MSG1-LENGTH PIC 99 VALUE 20.
01 ACK-MSG2-LENGTH PIC 99 VALUE 22.
01 ACK-MSG3-LENGTH PIC 99 VALUE 70.

COMMUNICATION SECTION.

```
CD IN-QUE
    FOR INITIAL INPUT.
01 IN-QUE-AREA.
    02 IN-QUES.
        04 IN-Q          PIC X(12).
        04 IN-SUBQ1     PIC X(12).
        04 IN-SUBQ2     PIC X(12).
        04 IN-SUBQ3     PIC X(12).
    02 IN-MSG-DATE     PIC 9(06).
    02 IN-MSG-TIME     PIC 9(08).
    02 IN-SYM-SOURCE   PIC X(12).
    02 IN-TXT-LENGTH   PIC 9(04).
    02 IN-END-KEY      PIC X.
    02 IN-STATUS-KEY   PIC XX.
    02 IN-MSG-COUNT    PIC 9(06).

CD OUT-QUE FOR OUTPUT.
01 OUT-QUE-AREA.
    02 OUT-DEST-COUNT  PIC 9(04)      VALUE 1.
    02 OUT-TXT-LENGTH  PIC 9(04).
    02 OUT-STATUS-KEY  PIC XX.
    02 OUT-DEST-ERKEY  PIC X.
    02 OUT-SYM-DEST    PIC X(12)     VALUE "CONSOL1".
```

PROCEDURE DIVISION.

```
ENABLE-MESSAGE.
    ENABLE OUTPUT OUT-QUE WITH KEY "COMMTST1".
    ACCEPT IN-QUE MESSAGE COUNT.
    PERFORM PROCESS-MESSAGES UNTIL IN-MSG-COUNT = 0.
    STOP RUN.
```

```
PROCESS-MESSAGES.
    RECEIVE IN-QUE MESSAGE INTO MSG-HOLD
        NO DATA
            MOVE NAK-MSG1-LENGTH TO OUT-TXT-LENGTH
            SEND OUT-QUE FROM NAK-MSG1 WITH EMI
            STOP RUN
    END-RECEIVE.
```

```
    MOVE ACK-MSG2-LENGTH TO OUT-TXT-LENGTH.
    SEND OUT-QUE FROM ACK-MSG2 WITH EMI.
```

```
    MOVE IN-QUES TO ACK-QUES.
    MOVE IN-SYM-SOURCE TO ACK-DEVICE.
    MOVE ACK-MSG3-LENGTH TO OUT-TXT-LENGTH.
    SEND OUT-QUE FROM ACK-MSG3 WITH EMI.
```

```
    MOVE IN-TXT-LENGTH TO OUT-TXT-LENGTH.
    SEND OUT-QUE FROM MSG-HOLD WITH EGI.
```

```
    ACCEPT IN-QUE MESSAGE COUNT.
```

Chapter 16. Segmentation

Contents

About This Chapter	16-3
Introduction	16-4
General Description of Segmentation	16-4
Organization	16-4
Segmentation Classification	16-5
Segmentation Control	16-5
Structure of Program Segments	16-6
Segment-Numbers	16-6
SEGMENT-LIMIT	16-7
Restrictions on Program Flow	16-7
ALTER Statement	16-7
PERFORM Statement	16-8
MERGE Statement	16-8
SORT Statement	16-8

About This Chapter

This chapter describes the Segmentation module and its technique to optimize storage use by allowing some Procedure Division sections to overlay one another during execution time.

Three types of segments are discussed in this chapter:

- Fixed permanent
- Fixed overlayable
- Independent

Introduction

The segmentation module provides a capability to specify object program overlay requirements.

Segmentation provides a facility for specifying permanent and independent segments. Segmentation also allows the intermixing of sections with different segment-numbers and allows the fixed portion of the source program to contain segments that may be overlaid.

General Description of Segmentation

COBOL segmentation enables a means for the user to communicate with the IBM AIX VS COBOL system to specify object program overlay requirements.

COBOL segmentation deals only with segmentation of procedures. As such, only the Procedure Division is considered in determining segmentation requirements for an object program.

Organization

This section is organized into Program segments, Fixed portion, and Independent segments.

Program Segments

Although it is not mandatory, the Procedure Division for a source program is usually written as a consecutive group of sections, each of which is composed of a series of closely related operations that are designed to collectively perform a particular function. However, when segmentation is used, the entire Procedure Division must be divided into sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

Fixed Portion

The fixed portion is defined as that part of the object program which is logically treated as if it were always in memory. This portion of the program is composed of fixed permanent segments, and fixed segments that can be overlaid.

A fixed permanent segment is a segment in the fixed portion which cannot be overlaid by any other part of the program.

A fixed overlayable segment is a segment in the fixed portion which, although logically treated as if it were always in memory, can be overlaid by another segment to optimize memory utilization. Variation of the number of fixed permanent segments in the fixed portion can be accomplished by using a special facility called the SEGMENT-LIMIT clause. Refer to "SEGMENT-LIMIT" on page 16-7. Such a segment, if called for by the program, is always made available in its last used state.

Independent Segments

An independent segment is defined as part of the object program which can overlay and can be overlaid by either a fixed overlayable segment or another independent segment. An independent segment is in its initial state whenever control is transferred (either implicitly or explicitly) to that segment for the first time during the execution of a program. On subsequent transfers of control to the segment, an independent segment is also in its initial state when:

1. Control is transferred to that segment as a result of the implicit transfer of control between consecutive statements from a segment with a different segment-number; that is, when control falls through into the independent segment from the physically preceding segment.
2. Control is transferred to that segment as the result of the implicit transfer of control between a SORT or MERGE statement, in a segment with a different segment-number, and an associated input or output procedure in that independent segment.
3. Control is transferred explicitly to that segment from a segment with a different segment-number (with the exception noted in 2).

On subsequent transfer of control to the segment, an independent segment is in its last-used state when:

1. Control is transferred implicitly to that segment from a segment with a different segment-number (except as noted in 1). This could happen, for example, when control is returned to the independent segment from a Declarative procedure.
2. Control is transferred explicitly to that segment as the result of the execution of an EXIT PROGRAM statement.

Segmentation Classification

Sections that are to be segmented are classified, using a system of segment-numbers and the following criteria:

1. **Logic requirements.** Sections that must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging either to one of the overlayable fixed segments or to one of the permanent segments. Sections that are used less frequently are normally classified as belonging to one of the independent segments, depending on logic requirements.
2. **Frequency of use.** Generally, the more frequently to which a section is referred, the lower its segment-number. The less frequently to which it is referred, the higher its segment-number.
3. **Relationship to other sections.** Sections that frequently communicate with one another should be given the same segment-numbers.

Segmentation Control

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. Control may be transferred within a source program to any paragraph in a section. It is not mandatory to transfer control to the beginning of a section.

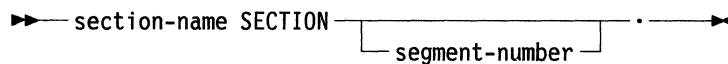
Structure of Program Segments

Segment-Numbers

Section classification is accomplished by means of a system of segment-numbers. The segment-number is included in the section header.

General Format

The following figure shows the format of the segment numbers:



Syntax Rules

The following syntax rules apply to segment-numbers:

1. Segment-number must be an integer ranging in value from 0 through 99.
2. If the segment-number is omitted from the section header, the segment-number is assumed to be 0.
3. Sections in the declaratives must contain segment numbers less than 50.

General Rules

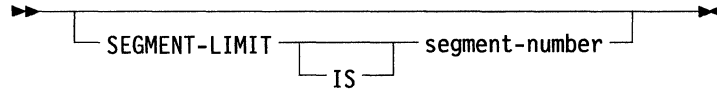
The following general rules apply to segment-numbers:

1. All sections which have the same segment-number constitute a program segment. All sections which have the same segment-number need not be physically contiguous in the source program.
2. Segments with segment-number 0 through 49 belong to the fixed portion of the object program.
3. Segments with segment-number 50 through 99 are independent segments.

SEGMENT-LIMIT

General Format

The SEGMENT-LIMIT clause appears in the OBJECT-COMPUTER paragraph and has the following format:



Syntax Rule

Segment-number must be an integer ranging in value from 1 through 49.

General Rule

The SEGMENT-LIMIT clause is for documentation purposes only.

Restrictions on Program Flow

When segmentation is used, the following restrictions are placed on the ALTER, PERFORM, MERGE and SORT statements.

ALTER Statement

A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

All other uses of the ALTER statement are valid and performed even if the GO TO to which the ALTER refers is in a fixed overlayable segment.

PERFORM Statement

A PERFORM statement that appears in a section that is not in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

- Sections and/or paragraphs wholly contained in one or more nonindependent segments.
- Sections and/or paragraphs wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

- Sections and/or paragraphs wholly contained in one or more nonindependent segments.
- Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

MERGE Statement

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 as either one or the other of the following:

- Totally within nonindependent segments
- 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 appear as either one or the other of the following:

- Totally within nonindependent segments
- Wholly within the same independent segment as that MERGE statement.

SORT Statement

If a SORT statement appears in a section that is not an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

- Totally within nonindependent segments, or
- Wholly contained in a single independent segment.

If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

- Totally within nonindependent segments, or
- Wholly within the same independent segment as that SORT statement.

Chapter 17. Program Debugging

Contents

About This Chapter	17-3
Introduction	17-4
Standard ANSI COBOL Debug	17-4
Environment Division in COBOL Debug	17-5
WITH DEBUGGING MODE Clause	17-5
Function	17-5
General Format	17-5
General Rules	17-5
Procedure Division in COBOL Debug	17-6
READY TRACE Statement	17-6
Function	17-6
General Format	17-6
General Rule	17-6
RESET TRACE Statement	17-7
Function	17-7
General Format	17-7
General Rule	17-7
USE FOR DEBUGGING Statement	17-8
Function	17-8
General Format	17-8
Syntax Rules	17-8
General Rules	17-9
Debugging Lines	17-13
Debugging Facilities Sample Program	17-14

About This Chapter

This chapter describes the AIX VS COBOL Debugging facility which provides the capability to generate program trace, to print debugging lines, or to monitor data items under predefined conditions.

Introduction

Debugging allows the user to describe the conditions under which procedures are to be monitored during the execution of the object program.

Standard ANSI COBOL Debug

The decisions of what to monitor and what information to display are explicitly in the domain of the user. The COBOL debug facility simply provides a convenient access to pertinent information.

The features of the language that support the COBOL debug module are:

- A WITH DEBUGGING MODE switch — used at object code creation time
- A run-time switch
- A USE FOR DEBUGGING statement
- A special register - DEBUG-ITEM
- Debugging lines.

The reserved word DEBUG-ITEM is the name for a special register generated automatically by the IBM AIX VS COBOL system that supports the debugging facility. Only one DEBUG-ITEM is allocated per program. The names of the subordinate data items in DEBUG-ITEM are also reserved words.

Compile-Time Switch

The DEBUGGING MODE clause is written as part of the SOURCE-COMPUTER paragraph in the Environment Division. It serves as a compile-time switch over debugging statements written in the source program.

When the WITH DEBUGGING MODE clause is specified in a program, all debugging sections and all debugging lines are handled as specified in this section of the document.

When DEBUGGING MODE is not specified in a program, all the debugging sections and debugging lines are treated as if they were comment lines and their syntax is not checked.

COBOL Debug Run-Time Switch

A run-time switch dynamically activates the debugging code inserted by the AIX VS COBOL system. This switch cannot be addressed in the program; it is controlled outside the COBOL environment. If the switch is on, the effects of any USE FOR DEBUGGING statements written in the source program are permitted. If the switch is off, all the effects described in the USE FOR DEBUGGING statements are inhibited. However, all debugging lines remain in effect. Therefore, you do not need to recompile in order to provide or remove this facility.

The run-time switch has no effect on the execution of the object program if the WITH DEBUGGING MODE clause was not specified in the source program.

The switch is described in the *User's Guide*.

Environment Division in COBOL Debug

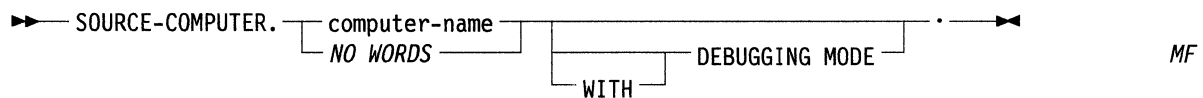
WITH DEBUGGING MODE Clause

Function

The WITH DEBUGGING MODE clause indicates that all debugging sections and all debugging lines are to be included in the object code. If this clause is not specified, all debugging lines and sections are treated as if they were comment lines.

General Format

The following figure shows the format of the WITH DEBUGGING MODE clause:



General Rules

The following general rules apply to the WITH DEBUGGING MODE clause:

1. If the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph of the CONFIGURATION SECTION of a program, all USE FOR DEBUGGING statements and all debugging lines are included in the object code.
2. If the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph of the CONFIGURATION SECTION of a program, any USE FOR DEBUGGING statements, all associated debugging sections, and any debugging lines are treated as if they were comment statements.

Procedure Division in COBOL Debug

READY TRACE Statement

Function

The READY TRACE statement is a debugging feature which causes each section and paragraph name to appear on the display screen in order of execution. OSVS VSC2
MF

General Format

The following figure shows the format of the Ready Trace statement:

►— READY TRACE —►

General Rule

The AIX VS COBOL system directive TRACE must be set to on to allow execution of the statement. READY TRACE is documentary only when the directive is not set. Refer to the User's Guide for details on setting the TRACE directive. OSVS VSC2
MF

RESET TRACE Statement

Function

The RESET TRACE statement disables the tracing of section and paragraph names on the display screen. OSVS VSC2
MF

General Format

The following figure shows the format of the RESET TRACE statement:

▶— RESET TRACE —▶

General Rule

The AIX VS COBOL system directive TRACE must be set to on to allow execution of the statement. RESET TRACE is documentary only when the directive is not set. Refer to the User's Guide for details on setting the TRACE directive. OSVS VSC2
MF

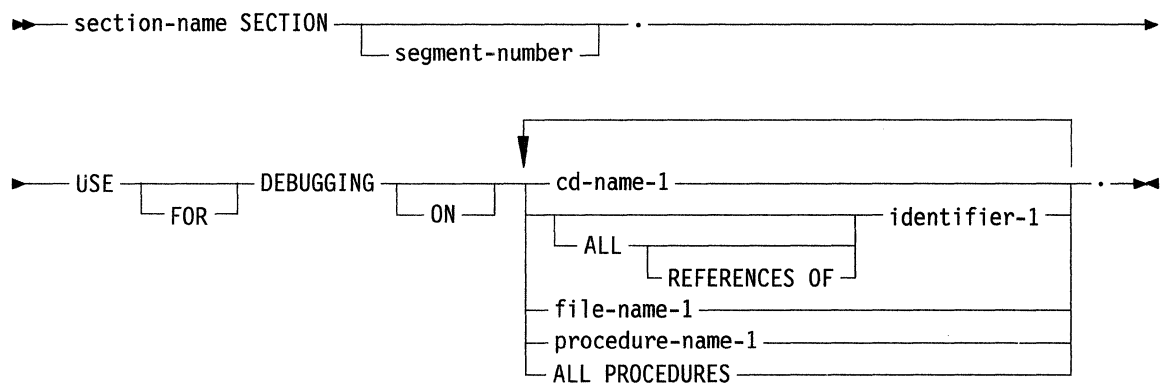
USE FOR DEBUGGING Statement

Function

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the debugging section.

General Format

The following figure shows the format of the USE FOR DEBUGGING statement:



Syntax Rules

The following syntax rules apply to the USE FOR DEBUGGING statement:

1. Debugging section(s), if specified, must appear together immediately after the Declaratives header.
2. Except in the USE FOR DEBUGGING statement itself, there must be no reference to any nondeclarative procedure within the debugging section.
3. Statements appearing outside of the set of debugging sections must not reference procedure-names defined within the set of debugging sections.
4. Except for the USE FOR DEBUGGING statement itself, statements appearing within a given debugging section may reference procedure-names defined within a different Use procedure only with a Perform statement.
5. Procedure-names defined within debugging sections must not appear within USE FOR DEBUGGING statements.
6. Any given identifier, cd-name, file-name, or procedure-name may appear in only one USE FOR DEBUGGING statement and may appear only once in that statement.
7. The ALL PROCEDURES phrase can appear only once in a program.
8. When the ALL PROCEDURES phrase is specified, procedure-name-1 must not be specified in any USE FOR DEBUGGING statement.

-
9. If the data description entry of the data item referenced by identifier-1 contains an OCCURS clause or is subordinate to a data description entry that contains an OCCURS clause, identifier-1 must be specified without the subscripting or indexing normally required.
 10. References to the special register DEBUG-ITEM are restricted to references from within a debugging section.

General Rules

The following general rules apply to the USE FOR DEBUGGING statement:

1. In the following general rules all references to cd-name, identifier-1, procedure-name-1, and file-name-1 apply equally to any additional instances of these items.
2. Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.
3. When file-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:
 - a. After the execution of any OPEN or CLOSE statement that references file-name-1, and
 - b. After the execution of any READ statement (after any other specified USE procedure) not resulting in the execution of an associated AT END or INVALID KEY imperative statement, and
 - c. After the execution of any DELETE or START statement that references file-name-1.
4. When procedure-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:
 - a. Immediately before each execution of the named procedure
 - b. Immediately after the execution of an ALTER statement that references procedure-name-1.
5. The ALL PROCEDURES phrase causes the effects described in 4 to occur for every procedure-name in the program, except those appearing within a debugging section.
6. When the ALL REFERENCES OF identifier-phrase is specified, that debugging section is executed for every statement that explicitly references identifier-1 at each of the following times:
 - a. In the case of a WRITE or REWRITE statement immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.
 - b. In the case of a GO TO statement with a DEPENDING ON phrase, immediately before control is transferred and prior to the execution of any debugging section associated with the procedure-name to which control is to be transferred.
 - c. In the case of a PERFORM statement in which a VARYING, AFTER, or UNTIL phrase references identifier-1, immediately after each initialization, modification or evaluation of the contents of the data item referenced by identifier-1.
 - d. In the case of any other COBOL statement, immediately after execution of that statement.

If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

-
7. When identifier-1 is specified without the ALL REFERENCES OF phrase, that debugging section is executed at each of the following times:
 - a. In the case of a WRITE or REWRITE statement that explicitly references identifier-1, immediately before the execution of that WRITE or REWRITE statement.
 - b. After the execution of any implicit move resulting from the presence of the FROM phrase.
 - c. In the case of a PERFORM statement in which a VARYING, AFTER, or UNTIL phrase references identifier-1, immediately after each initialization, modification or evaluation of the contents of the data item referenced by identifier-1.
 - d. Immediately after the execution of any other COBOL statement that explicitly references and causes the contents of the data item referenced by identifier-1 to be changed.

If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

8. The associated debugging section is not executed for a specific operand more than once as a result of the execution of a single statement, regardless of the number of times that operand is explicitly specified. In the case of a PERFORM statement which caused iterative execution of a referenced procedure, the associated debugging section is executed once for each iteration.

Within an imperative statement, each individual occurrence of an imperative verb identifies a separate statement for the purpose of debugging.

9. When cd-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:
 - a. After the execution of any ENABLE, DISABLE, and SEND statement that references cd-name-1.
 - b. After the execution of a RECEIVE statement referencing cd-name-1 that does not result in the execution of the NO DATA imperative-statement, and
 - c. After the execution of an ACCEPT MESSAGE COUNT statement that references cd-name-1.
10. A reference to file-name-1, identifier-1, procedure-name-1, or cd-name-1 as a qualifier does not constitute reference to that item for the debugging described in the general rules above.
11. Associated with each execution of a debugging section is the special register DEBUG-ITEM, which provides information about the conditions that caused the execution of a debugging section. DEBUG-ITEM has the following implicit description:

```

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

```

-
12. Prior to each execution of a debugging section, the contents of the data item referenced by DEBUG-ITEM are space-filled. According to the following general rules, the contents of data items subordinate to DEBUG-ITEM are then updated immediately before control is passed to that debugging section. Spaces remain as the contents of any data item not specified in the general rules that follow.

Updating is accomplished in accordance with the rules for the MOVE statement, the sole exception being the move to DEBUG-CONTENTS when the move is treated exactly as if it was an alphanumeric to alphanumeric elementary move with no conversion of data from one form of internal representation to another.

13. The contents of DEBUG-LINE is the relevant COBOL source line number. This provides the means of identifying a particular source statement.
14. DEBUG-NAME contains the first 30 characters of the name that caused the debugging section to be executed.

All qualifiers of the name are separated in DEBUG-NAME by the word IN or OF.

Subscripts/indexes, if any, are not entered into DEBUG-NAME.

15. If the reference to a data item that causes the debugging section to be executed is subscripted or indexed, the occurrence number of each level is entered, as necessary, in DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3, respectively.
16. DEBUG-CONTENTS is a data item that is large enough to contain the data required by the general rules that follow.
17. If execution of the first nondeclarative procedure in the program causes the debugging section to be executed, the following conditions exist:
 - a. DEBUG-LINE identifies the first statement of that procedure.
 - b. DEBUG-NAME contains the name of that procedure.
 - c. DEBUG-CONTENTS contains START PROGRAM.
18. If a reference to procedure-name-1 in an ALTER statement causes the debugging section to be executed, the following conditions exist:
 - a. DEBUG-LINE identifies the ALTER statement that references procedure-name-1.
 - b. DEBUG-NAME contains procedure-name-1.
 - c. DEBUG-CONTENTS contains the applicable procedure-name associated with the TO phrase of the ALTER statement.
19. If the transfer of control associated with the execution of a GO TO statement causes the debugging section to be executed, the following conditions exist:
 - a. DEBUG-LINE identifies the GO TO statement whose execution transfers control to procedure-name-1.
 - b. DEBUG-NAME contains procedure-name-1.
20. If reference to procedure-name-1 in the INPUT or OUTPUT phrase of a SORT or MERGE statement causes the debugging section to be executed, the following conditions exist:
 - a. DEBUG-LINE identifies the SORT or MERGE statement that references procedure-name-1.
 - b. DEBUG-NAME contains procedure-name-1.
 - c. DEBUG-CONTENTS contains:
 - 1) SORT INPUT, if the reference to procedure-name-1 is the INPUT phrase of a SORT statement.
 - 2) SORT OUTPUT, if the reference to procedure-name-1 is in the OUTPUT phrase of a SORT statement.
 - 3) MERGE OUTPUT, if the reference to procedure-name-1 is in the OUTPUT phrase of a MERGE statement.

-
21. If the transfer of control from the control mechanism associated with a **PERFORM** statement causes the debugging section associated with `procedure-name-1` to be executed, the following conditions exist:
 - a. **DEBUG-LINE** identifies the **PERFORM** statement that references `procedure-name-1`.
 - b. **DEBUG-NAME** contains `procedure-name-1`.
 - c. **DEBUG-CONTENTS** contains **PERFORM LOOP**.
 22. If `procedure-name-1` is a **USE** procedure that is to be executed, the following conditions exist:
 - a. **DEBUG-LINE** identifies the statement that causes execution of the **USE** procedure.
 - b. **DEBUG-NAME** contains `procedure-name-1`.
 - c. **DEBUG-CONTENTS** contains **USE PROCEDURE**.
 23. If an implicit transfer of control from the previous sequential paragraph to `procedure-name-1` causes the debugging section to be executed, the following conditions exist:
 - a. **DEBUG-LINE** identifies the previous statement.
 - b. **DEBUG-NAME** contains `procedure-name-1`.
 - c. **DEBUG-CONTENTS** contains **FALL THROUGH**.
 24. If references to `file-name-1`, `cd-name-1` causes the debugging section to be executed, then:
 - a. **DEBUG-LINE** identifies the source statement that references `file-name-1`, `cd-name-1`.
 - b. **DEBUG-NAME** contains the name of `file-name-1`, `cd-name-1`.
 - c. For **READ**, **DEBUG-CONTENTS** contains the entire record read.
 - d. For all other references to `file-name-1`, **DEBUG-CONTENTS** contains spaces.
 - e. For any reference `cd-name-1`, **DEBUG-CONTENTS** contains the contents of the area associated with the `cd-name`.
 25. If a reference to `identifier-1` causes the debugging section to be executed, then:
 - a. **DEBUG-LINE** identifies the source statement that references `identifier-1`.
 - b. **DEBUG-NAME** contains the name of `identifier-1`.
 - c. **DEBUG-CONTENTS** contains the contents of the data item referenced by `identifier-1` at the time that control passes to the debugging section. Refer to rules 6 and 7 on page 17-10.

Debugging Lines

A debugging line is any line with a D in the indicator area of the line. Any debugging line that consists solely of spaces from margin A to margin R is considered the same as a blank line.

The contents of a debugging line must allow a syntactically correct program to be formed with or without the debugging lines being considered as comment lines.

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.

Successive debugging lines are allowed. Continuation of debugging lines is permitted, except that each continuation line must contain a D in the indicator area and character strings may not be broken across two lines.

A debugging line is only permitted in the program after the OBJECT-COMPUTER paragraph.

Debugging Facilities Sample Program

The following example shows a COBOL program skeleton with debugging facilities:

```
IDENTIFICATION DIVISION.  
  :  
  :  
CONFIGURATION SECTION.  
SOURCE-COMPUTER.  IBM-RISC-6000 WITH DEBUGGING MODE.  
  :  
  :  
PROCEDURE DIVISION.  
DECLARATIVES.  
  :  
DEBUG1 SECTION.  USE FOR DEBUGGING ON UPDATE-PAY-MASTER.  
DEBUG-11.  
  PERFORM SHOW-DEBUG-ITEMS.  
  PERFORM SHOW-PAY-MASTER-INFO.  
DEBUG-12.  
  EXIT.  
  
DEBUG2 SECTION.  USE FOR DEBUGGING ON ALL REFERENCES OF TOTAL-AMOUNT.  
DEBUG-21.  
  PERFORM SHOW-DEBUG-ITEM.  
  PERFORM SHOW-DATA-TOTAL-AMT.  
DEBUG-22.  
  EXIT.  
  :  
END DECLARATIVES.  
  
PAYROLL-MONTHLY SECTION.  
  :  
  READY TRACE.  
UPDATE-PAY-MASTER.  
  :  
DELETE-PAY-MASTER.  
  :  
  RESET TRACE.  
  :  
  :  
  
* The next two statements are debugging lines. They will be  
* considered as comments if the WITH DEBUGGING MODE is not specified.  
  
D DISPLAY "This is a debugging line".  
D DISPLAY "Another debugging line".  
  :  
  :  
  :
```

Chapter 18. Screen-Handling

Contents

About This Chapter	18-5
Introduction	18-6
Environment Division in the Screen-Handling Module	18-8
SPECIAL-NAMES Paragraph	18-9
Function	18-9
General Format	18-9
CONSOLE IS CRT Clause	18-10
Function	18-10
General Format	18-10
General Rule	18-10
CURSOR IS Clause	18-11
Function	18-11
General Format	18-11
Syntax Rule	18-11
General Rules	18-11
CRT STATUS Clause	18-12
Function	18-12
General Format	18-12
Syntax Rule	18-12
General Rule	18-12
Example	18-13
Data Division in the Screen-Handling Module	18-14
SCREEN SECTION	18-14
Function	18-14
General Format	18-14
Syntax Rules	18-14
General Rule	18-14
Screen Description — Complete Entry Skeleton	18-15
Function	18-15
General Format	18-16
Syntax Rules	18-17
General Rules	18-18
Example 1	18-19
Example 2	18-19
AUTO Clause	18-21
Function	18-21
General Format	18-21
Syntax Rules	18-21
General Rules	18-21
BACKGROUND-COLOR Clause	18-22
Function	18-22
General Format	18-22
Syntax Rules	18-22
General Rules	18-22
BELL Clause	18-24
Function	18-24
General Format	18-24
Syntax Rules	18-24
BLANK Clause	18-25
Function	18-25
General Format	18-25
Syntax Rule	18-25
General Rules	18-25
Example	18-25
BLANK WHEN ZERO Clause	18-26
Function	18-26
General Format	18-26
Syntax Rule	18-26
BLINK Clause	18-27

Function	18-27
General Format	18-27
Syntax Rules	18-27
COLUMN Clause	18-28
Function	18-28
General Format	18-28
Syntax Rules	18-28
General Rules	18-28
FOREGROUND-COLOR Clause	18-30
Function	18-30
General Format	18-30
Syntax Rules	18-30
General Rules	18-30
Example	18-31
FULL Clause	18-32
Function	18-32
General Format	18-32
Syntax Rules	18-32
General Rules	18-32
GRID Clause	18-34
Function	18-34
General Format	18-34
Syntax Rules	18-34
HIGHLIGHT Clause	18-35
Function	18-35
General Format	18-35
Syntax Rules	18-35
General Rule	18-35
Example	18-35
JUSTIFIED Clause	18-36
Function	18-36
General Format	18-36
Syntax Rules	18-36
General Rules	18-36
LEFTLINE Clause	18-37
Function	18-37
General Format	18-37
Syntax Rules	18-37
LINE Clause	18-38
Function	18-38
General Format	18-38
Syntax Rules	18-38
General Rules	18-38
OCCURS Clause	18-40
Function	18-40
General Format	18-40
Syntax Rules	18-40
General Rules	18-40
Example	18-41
OVERLINE Clause	18-42
Function	18-42
General Format	18-42
Syntax Rules	18-42
PICTURE Clause	18-43
Function	18-43
General Format	18-43
Syntax Rules	18-43
General Rules	18-43
PROMPT Clause	18-45
Function	18-45
General Format	18-45
Syntax Rules	18-45

General Rules	18-45
REQUIRED Clause	18-46
Function	18-46
General Format	18-46
Syntax Rules	18-46
General Rules	18-46
REVERSE-VIDEO Clause	18-47
Function	18-47
General Format	18-47
General Rules	18-47
SECURE Clause	18-48
Function	18-48
General Format	18-48
Syntax Rules	18-48
General Rule	18-48
Example	18-48
SIGN Clause	18-49
Function	18-49
General Format	18-49
Syntax Rules	18-49
General Rule	18-49
SIZE Clause	18-50
Function	18-50
General Format	18-50
Syntax Rules	18-50
General Rules	18-50
UNDERLINE Clause	18-51
Function	18-51
General Format	18-51
Syntax Rules	18-51
VALUE Clause	18-52
Function	18-52
General Format	18-52
Syntax Rules	18-52
ZERO-FILL Clause	18-53
Function	18-53
General Format	18-53
Syntax Rule	18-53
General Rules	18-53
Procedure Division	18-54
ACCEPT Statement	18-55
Function	18-55
General Format	18-55
Syntax Rules	18-57
General Rules	18-58
DISPLAY Statement	18-61
Function	18-61
General Format	18-61
Syntax Rules	18-62
General Rules	18-63
Screen-Handling Sample Program	18-65

About This Chapter

This chapter describes the AIX VS COBOL Screen-Handling module and describes how it supports formatted full-screen output and input using enhanced DISPLAY/ACCEPT statement and additional statements in the Environment Division and Data Division.

Introduction

Note: This material in this chapter pertains exclusively to the Micro Focus dialect.

The Screen-Handling module provides the user with enhanced screen-handling facilities. It consists of a SCREEN SECTION and additional formats of ACCEPT and DISPLAY statements.

Two methods of screen input-output operations are available:

1. The display of nonscrolling forms consisting of areas of the screen as defined in the SCREEN SECTION. A SCREEN SECTION entry is a *screen description*. It is similar in appearance to a data description but defines a *screen item* or area of the display screen rather than an area in memory.
2. The display of data items which constitute nonscrolling forms. The details of the areas of the display screen to be used are provided in the associated ACCEPT and DISPLAY operations.

The SCREEN SECTION contains a description of each field on the display screen that is accessed in a Format 1 ACCEPT or DISPLAY operation. Such a field is called a screen item. Many screen items describe only the layout of fields within a field on the display screen and are never referenced explicitly.

The SCREEN SECTION contains syntax which enables the operator to:

- Specify the exact location of fields.
- Accept data typed at specified positions.
- Display literal text at specified positions.
- Define screen attributes.
- Control console features.

Details of how a screen item is to look are given by screen description clauses in its description. Details of how a data item is to look on the display screen are given by *screen options* in the WITH phrase of the ACCEPT or DISPLAY statement. Many screen options are the same as screen description clauses.

Figure 18-1 on page 18-7 summarizes the screen description clauses, screen options, and data description clauses available in the SCREEN SECTION and for use with ACCEPT and DISPLAY statements. It also specifies options or clauses that can be used with each of the types of fields defined below:

- An input field is a screen item whose description contains a TO phrase.
- An output field is a screen item whose description contains a FROM phrase.
- An update field is a screen item whose description contains a USING phrase.
- A literal field is an elementary screen item whose description contains no PICTURE clause.

There are two ways to enter data into numeric and numeric-edited screen fields: fixed format mode and free format mode.

Fixed format mode is the default manner in which data entry is made to numeric and numeric-edited screen fields. This mode formats and echoes the entered data and also moves the cursor in accordance with the requirements of the field's picture specification, as each keystroke is received. Characters other than +, -, and the decimal point character are rejected; insertion characters in edited fields are skipped over as the cursor moves backwards and forwards; any sign indicator is modified in accordance with its normal specification; floating symbols move left and right in the field and insertion symbols are inserted or deleted as digits.

Free format mode is an alternative manner in which data entry can be assigned to numeric and numeric-edited screen fields. The default mode is fixed format mode (see above entry). This configurable mode allows data to be keyed into a PIC X field of appropriate length, and it is only when the operator leaves the field that the data is reformatted to comply with the picture specification. Once the operator moves the cursor from the field, the IBM AIX VS COBOL system disregards all characters other than digits and the sign and

decimal point symbols. It then extracts, stores, or reformats the numeric value in accordance with the normal COBOL rules for a MOVE to an item with the same picture as the screen or working-storage item. The numeric value is then usually echoed to the screen.

Screen Clauses/ Screen Options/ Data Description Clauses	SCREEN SECTION				WITH PHRASE	
	Input Field	Output Field	Update Field	Literal Field	ACCEPT	DISPLAY
AUTO	X		X		X	
BACKGROUND-COLOR	X	X	X	X	X	X
BELL	X	X	X	X	X	X
BLANK	X	X	X	X		X
BLANK WHEN ZERO ¹	X	X	X			
BLINK	X	X	X	X	X	X
COLUMN	X	X	X	X		
FOREGROUND-COLOR	X	X	X	X	X	X
FULL	X		X		X	
GRID	X	X	X	X	X	X
HIGHLIGHT	X	X	X	X	X	X
JUSTIFIED ¹	X	X	X			
LEFT-JUSTIFY					X	
LEFTLINE	X	X	X	X	X	X
LINE	X	X	X	X		
OCCURS ¹	X	X	X			
OVERLINE	X	X	X	X	X	X
PROMPT	X		X		X	
REQUIRED	X		X		X	
REVERSE-VIDEO	X	X	X	X	X	X
RIGHT-JUSTIFY					X	
SECURE	X		X		X	
SIGN ¹	X	X	X			
SIZE	X	X	X	X	X	X
SPACE-FILL					X	
TRAILING-SIGN					X	
UNDERLINE	X	X	X	X	X	X
UPDATE					X	
ZERO-FILL	X		X		X	

X = clause/option allowed

¹ = Data description clause allowed in the SCREEN SECTION

Figure 18-1. Permitted Use Of Options

These clauses and options are described in the following sections.

Note: The attributes GRID, LEFTLINE, and OVERLINE are provided for compatibility. They are accepted syntactically but have no effect at run time. The attribute BLINK is always accepted syntactically but will have effect only on terminals that support blinking.

Environment Division in the Screen-Handling Module

The Environment Division in the Screen-Handling Module contains the following sections:

- SPECIAL-NAMES Paragraph
- CONSOLE IS CRT Clause
- CURSOR IS Clause
- CRT STATUS Clause.

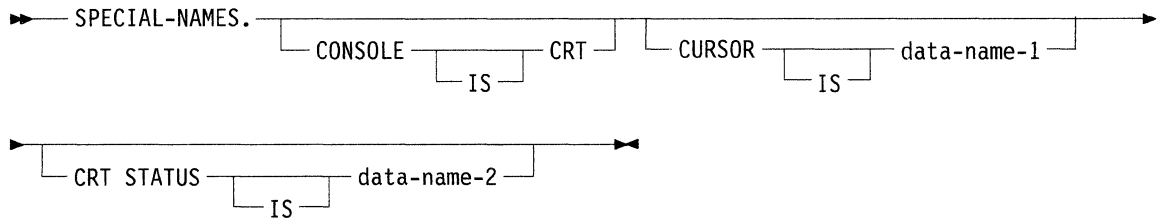
SPECIAL-NAMES Paragraph

Function

The SPECIAL-NAMES paragraph provides a means for relating implementer names to user-specified mnemonic names.

General Format

The following figure shows the format of the SPECIAL-NAMES paragraph:



Syntax rules and general rules for the SPECIAL-NAMES paragraph are listed below each clause.

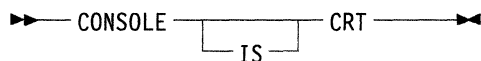
CONSOLE IS CRT Clause

Function

The CONSOLE IS CRT clause changes the default interpretation of ACCEPT or DISPLAY statements from the ANSI standard format to Format 2 described later in this chapter.

General Format

The following figure shows the format of the CONSOLE IS CRT clause:



General Rule

The CONSOLE IS CRT clause causes any ACCEPT or DISPLAY statement whose operand is not a screen-name, and that has no phrases specific to a particular format, to be treated as Format 2. If the CONSOLE IS CRT clause is not present, these statements are treated as the standard ANSI ACCEPT or DISPLAY. Refer to “ACCEPT Statement” on page 7-22, “ACCEPT Statement” on page 18-55, “DISPLAY Statement” on page 18-61, and “DISPLAY Statement” on page 7-32.

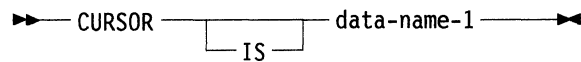
CURSOR IS Clause

Function

The CURSOR IS clause specifies the data item to contain the cursor address as used by the ACCEPT statement.

General Format

The following figure shows the format of the CURSOR IS clause:



Syntax Rule

data-name-1 must be declared in the WORKING-STORAGE SECTION of the program.

General Rules

The following general rules apply to the CURSOR IS clause:

1. At the start of an ACCEPT statement, if data-name-1 contains a value that is a valid character position on the display screen (see rule 4), that position is used as the initial position for the cursor. Otherwise, data-name-1 is ignored, and the initial position for the cursor is the start of the first input field on the screen. At the end of an ACCEPT statement, if the position in data-name-1 has been used in that statement, data-name-1 is updated to show the position of the cursor at the termination of the ACCEPT statement.
2. CURSOR IS has no effect on the positioning of fields on the screen.
3. data-name-1 must be 4 or 6 characters in length. If data-name-1 is 4 characters in length, the first two characters are interpreted as line number, and the second two as column number. If data-name-1 is 6 characters in length, the first three characters are interpreted as line number, and the second three as column number.
4. The clause has no effect if data-name-1 contains an illegal position (for example, zeros, a nonnumeric value, or a value that is beyond the bottom of the screen).
5. If data-name-1 contains a valid position that does not correspond to an input field being accepted by the current ACCEPT statement, the cursor is positioned to the next such field or, if there is none, to the first such field. The ordering of the fields is the order their descriptions appear in the Data Division.

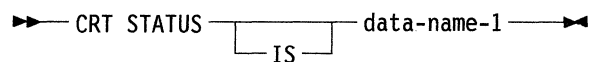
CRT STATUS Clause

Function

The CRT STATUS clause specifies a data item into which a status value is moved after each Format 1 or 2 ACCEPT statement.

General Format

The following figure shows the format of the CRT STATUS clause:



Syntax Rule

data-name-1 must be described in the WORKING-STORAGE SECTION and must be three bytes long.

General Rule

If the CRT STATUS clause is specified in the SPECIAL-NAMES paragraph, every Format 1 or 2 ACCEPT statement (as described later in this chapter) places a value into data-name-1 to indicate the outcome of the ACCEPT operation. data-name-1 consists of status keys which are set to indicate possible conditions resulting from the completion of the operation. They are described below.

CRT Status Key 1

The first byte of data-name-1 is CRT Status Key 1 and should be described as PICTURE 9 USAGE DISPLAY. It indicates the condition that caused the termination of the ACCEPT operation. The possible values are:

- 0 Indicates a terminator key or autoskip out of the final field
- 1 Indicates a user-defined function key
- 2 Indicates an AIX VS COBOL defined function key
- 9 Indicates an error

A **terminator key** is a key whose purpose is terminating ACCEPT operations (for example, send). There is a configuration option that causes the field-tab key, when used in the final field of an ACCEPT, to act as a terminator key also. Defining function keys is also a configuration option.

A termination that returns a value of 0 is a **normal termination**.

If the ACCEPT statement contains an ON EXCEPTION phrase, any value in CRT Status Key 1, except 0, will cause the execution of the imperative-statement in that phrase.

CRT Status Key 2

The second byte of data-name-1 is CRT Status Key 2 and contains a code giving further details of the condition that terminated the ACCEPT operation. Its format and possible values depend on the value in CRT Status Key 1, as shown in Table 18-1.

KEY 1	KEY 2		Meaning
	Format	Value	
0	PIC 9 DISPLAY	0	The operator pressed a terminator key.
0	PIC 9 DISPLAY	1	Autoskip out of the last field.
1	PIC 99 COMP	0-127	The user-defined function key number.
2	PIC 99 COMP	0-26	The ADIS function key number.
9	PIC 99 COMP	0	No items fall within the screen.

Refer to the *User's Guide* for an explanation of function key numbers.

CRT Status Key 3

The third byte of data-name-1 is CRT Status Key 3. If CRT Status Key 1 and CRT Status Key 2 are zero, then CRT Status Key 3 contains the raw keyboard code for the key that terminated the ACCEPT operation. Otherwise, the contents of CRT Status Key 3 are undefined.

Where a sequence of keystrokes rather than a single key has been configured to perform a single function, only the code for the first keystroke is returned.

The following is an example of the CRT STATUS clause:

Example

```
ENVIRONMENT DIVISION.  
    :  
    :  
SPECIAL-NAMES.  
    CONSOLE IS CRT  
    CURSOR IS CURSOR-Y-X  
    CRT STATUS IS CRT-STATUS.  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
77  CRT-STATUS  PIC  X(3).  
01  CURSOR-Y-X.  
    02  CUR-Y    PIC  99.  
    02  CUR-X    PIC  99.  
    :  
    :
```

Data Division in the Screen-Handling Module

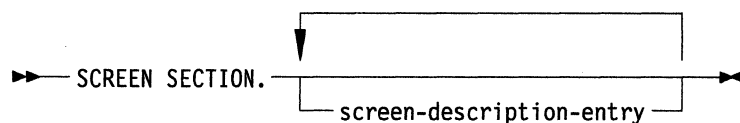
SCREEN SECTION

Function

The SCREEN SECTION provides screen-handling facilities for use with Format 1 ACCEPT and DISPLAY statements.

General Format

The following figure shows the format of the SCREEN SECTION:



Syntax Rules

The following syntax rules apply to the SCREEN SECTION:

1. The SCREEN SECTION must be the last section in the Data Division.
2. Screen-description-entries contain screen description clauses. Data description clauses allowed in the SCREEN SECTION are restricted to those described in this chapter.

General Rule

The SCREEN SECTION contains a description of each field on the screen that is accessed in a Format 1 ACCEPT or DISPLAY operation. Such a field is called a screen item. Many screen items describe only the layout of fields within a field on the display screen and are never referenced explicitly.

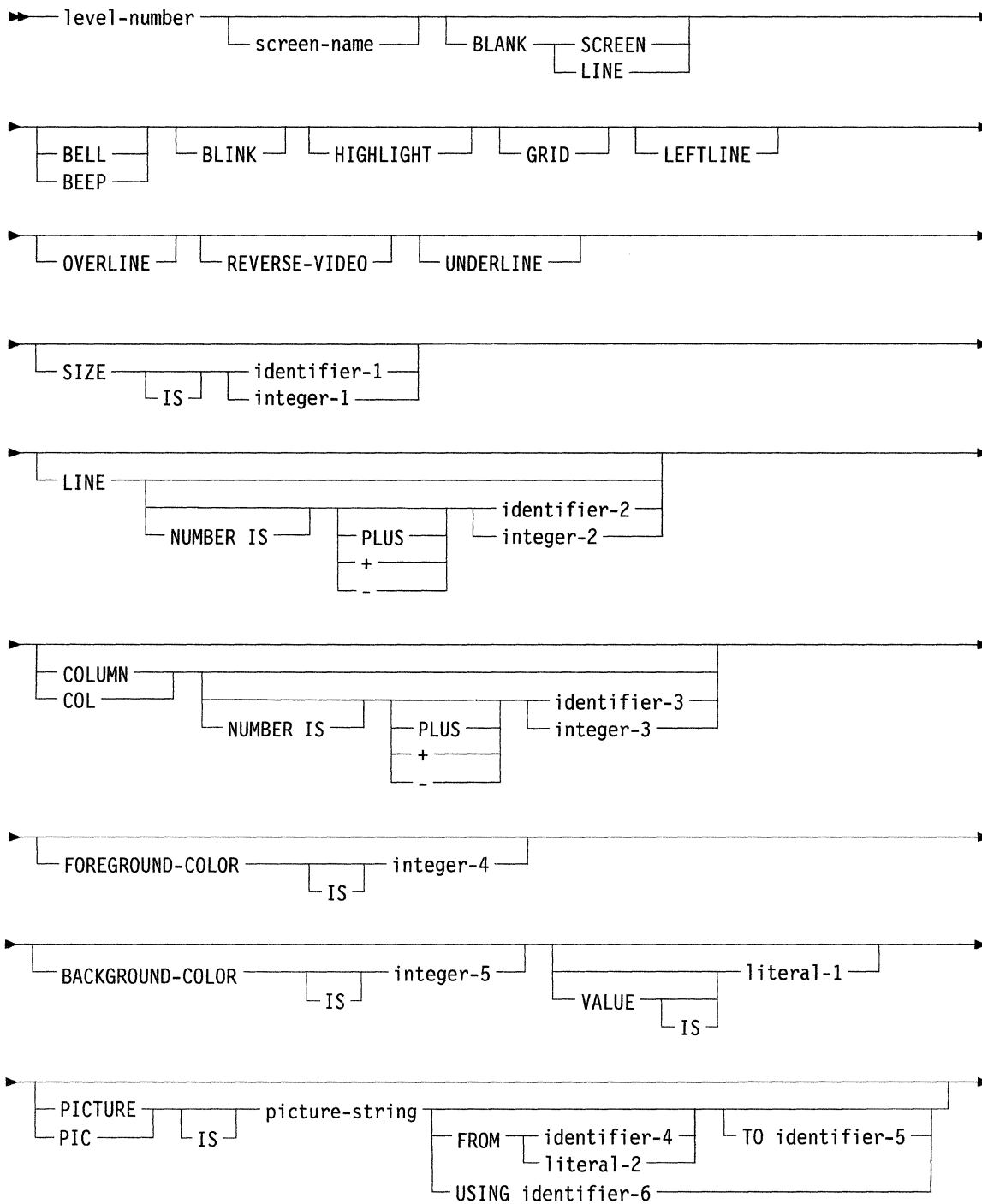
Screen Description – Complete Entry Skeleton

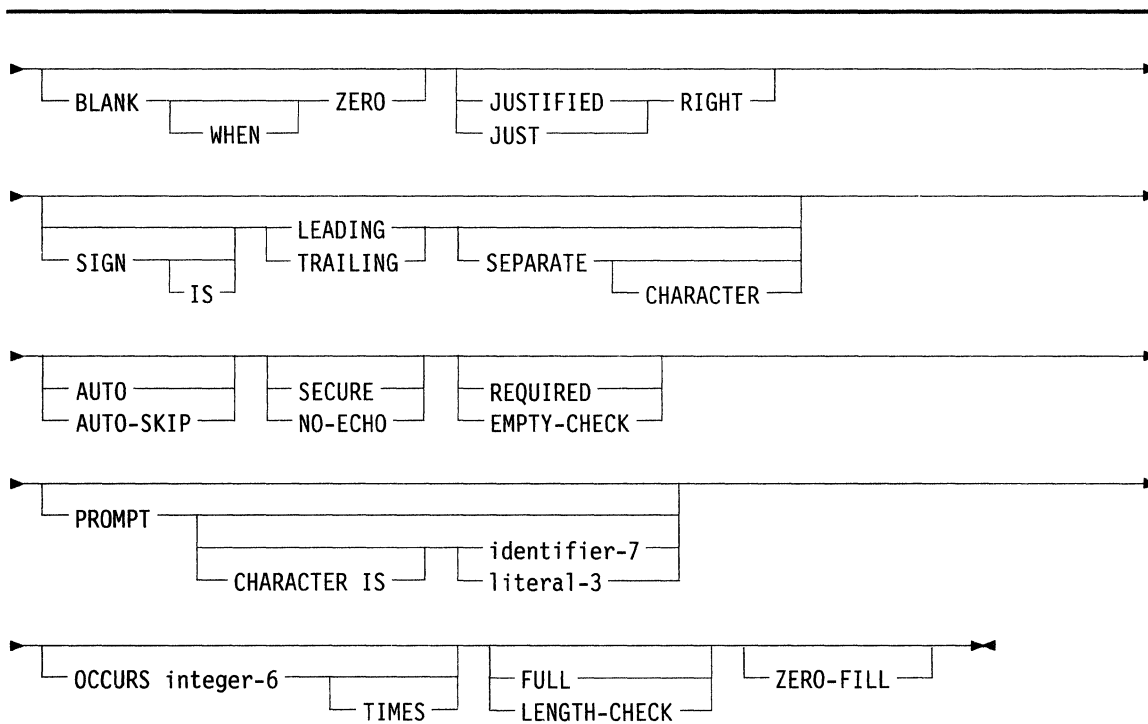
Function

A screen description entry specifies the attributes, behavior, size and location for a referenced screen item that is accepted or displayed at run time.

General Format

The following figure shows the general format of the Screen Description — Complete Entry Skeleton:





Syntax Rules

The following syntax rules apply to the screen description complete entry skeleton:

1. Each screen description entry must start with a level number from 01 through 49. Refer to Chapter 3, "Introduction to the Nucleus" for further details on level numbers.
2. Each level 01 item must have a screen-name.
3. Screen-name assigns a name to the screen item described in the screen description. Screen-name immediately follows level number, conforming to the rules for user-defined names.
4. A screen item can only be referenced in a Format 1 ACCEPT or DISPLAY statement.
5. Each elementary screen item must contain at least one of the following clauses: BELL, BLANK LINE, BLANK SCREEN, COLUMN, LINE, PICTURE, and VALUE.
6. The data items in the FROM, TO, and USING phrases are associated with the screen item. The USING phrase is equivalent to the combination of a FROM and TO phrase, each specifying the same field.
7. An ACCEPT statement can be executed on a group screen item containing screen items with FROM or VALUE phrases only if that group also contains screen items with TO or USING phrases.
8. The clauses following screen-name can be specified in any order.
9. A clause that appears in the description of a group screen item applies to all the elementary subordinate items in that group in whose descriptions it would be allowed.
10. The word FILLER is not permitted in screen description but the screen-name can be omitted provided the level-number is not 01.

-
11. If the same clause is specified more than once for the same screen item, the clause that appears at the lowest level within the hierarchy is the one which takes effect.
 12. The attributes GRID, LEFTLINE, and OVERLINE are provided for compatibility. They are accepted syntactically but have no effect at run time. The attribute BLINK is always accepted syntactically but will have effect only on terminals that support blinking.

General Rules

The following general rules apply to the screen description complete entry skeleton:

1. Screen descriptions define areas on the screen. Each entry consists of a level number, an optional screen-name, and various optional clauses relating to the positioning of fields as well as to console functions.
2. When the screen item is displayed, data is taken from the literal or data item named in the associated FROM or USING phrase. Items within the TO phrase only are treated as though FROM SPACE or FROM ZERO were specified, depending on the type of screen item.
3. When the screen item is accepted, the data entered is moved from the display screen to the data item named in the TO or USING phrase. Depending on the category of the item, conversion and de-editing are done, if necessary.
4. An *input field* is a screen item whose description contains a TO phrase.
5. An *output field* is a screen item whose description contains a FROM phrase.
6. An *update field* is a screen item whose description contains a USING phrase.
7. A *literal field* is an elementary screen item whose description contains no PICTURE clause.
8. An ACCEPT of a group screen item consists of accepting those elementary subordinate items that are input or update fields. They are accepted in the order their descriptions appear in the SCREEN SECTION at the display screen positions indicated by the screen descriptions. Unless otherwise specified in the CURSOR IS clause, the cursor is initially positioned at the start of the first item. Refer to "CURSOR IS Clause" on page 18-11. As the ACCEPT operation into each item is terminated, the cursor moves to the start of the next item.
9. A DISPLAY of a group screen item consists of displaying those elementary subordinate items that are output, update, or literal fields. They are displayed simultaneously at the screen positions indicated by the screen descriptions.
10. If the length of an ACCEPT or DISPLAY screen item exceeds the length of the current line, wraparound is to the next line.
11. If a screen item is too large to fit within the physical screen, truncation occurs at the first character that is off-screen for output fields and alphanumeric input and update fields, and at the first field that is off-screen for numeric and numeric-edited input and update fields.

Example 1

The following example shows the screen description entry:

```
01 TITLES-SCREEN.
   05 BLANK SCREEN.
   05 LINE 3 HIGHLIGHT
      VALUE "SCREEN PICTURE".
   05 COL 30 HIGHLIGHT
      VALUE "W-S PICTURE".
   05 COL 60 HIGHLIGHT
      VALUE "INPUT FIELD".
   05 LINE 15 COL 20 HIGHLIGHT
      VALUE "RECEIVING FIELDS".
   05 LINE REVERSE-VIDEO VALUE "FIELD-1=".
   05 LINE REVERSE-VIDEO VALUE "FIELD-2=".
   05 LINE REVERSE-VIDEO VALUE "FIELD-3=".
   05 LINE REVERSE-VIDEO VALUE "FIELD-4=".
```

Example 2

The following example shows another screen description entry:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.    CONSOLE IS CRT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 EMPLOYEE-RECORD.
   05 E-NAME          PIC X(25).
   05 E-ADDRESS       PIC X(35).
   05 E-CITY          PIC X(15).
   05 E-STATE         PIC X(02).
   05 E-ZIP-CODE      PIC X(05).

77 TITLE-LINE        PIC X(20) VALUE "Employee Data Screen".
77 NAME-LINE         PIC X(14) VALUE "Employee Name:".
77 ADDRESS-LINE      PIC X(08) VALUE "Address:".
77 CITY-LINE         PIC X(05) VALUE "City:".
77 STATE-LINE        PIC X(06) VALUE "State:".
77 ZIP-CODE-LINE     PIC X(09) VALUE "Zip Code:".
```

SCREEN SECTION.

01 DATA-SCREEN.

05 BLANK SCREEN.

05 LINE 1 COLUMN 30 PIC X(20) FROM TITLE-LINE.

05 LINE 3 COLUMN 5 PIC X(14) FROM NAME-LINE.

05 LINE 3 COLUMN 22 PIC X(25) TO E-NAME.

05 LINE 5 COLUMN 5 PIC X(08) FROM ADDRESS-LINE.

05 LINE 5 COLUMN 22 PIC X(35) TO E-ADDRESS.

05 LINE 7 COLUMN 5 PIC X(05) FROM CITY-LINE.

05 LINE 7 COLUMN 22 PIC X(15) TO E-CITY.

05 LINE 9 COLUMN 5 PIC X(06) FROM STATE-LINE.

05 LINE 9 COLUMN 22 PIC X(02) TO E-STATE.

05 LINE 11 COLUMN 5 PIC X(09) FROM ZIP-CODE-LINE.

05 LINE 11 COLUMN 22 PIC X(05) TO E-ZIP-CODE.

PROCEDURE DIVISION.

SR1.

DISPLAY SPACE.

DISPLAY DATA-SCREEN.

ACCEPT DATA-SCREEN.

:

:

STOP RUN.

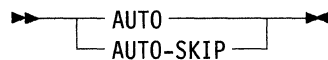
AUTO Clause

Function

The AUTO clause automatically terminates an ACCEPT operation of the screen item when the last character position is keyed. No explicit terminator key is necessary.

General Format

The following figure shows the format of the AUTO clause:



Syntax Rules

The following syntax rules apply to the AUTO clause:

1. AUTO and AUTO-SKIP are synonymous.
2. The AUTO clause is allowed only with input and update fields.
3. If this clause is specified at group level, it applies to all elementary subordinate items.

General Rules

The following general rules apply to the AUTO clause:

1. Provided any REQUIRED or FULL clause is satisfied, the cursor is positioned to the next screen item. Alternatively, if the screen item is the last one in the ACCEPT operation, the entire ACCEPT is terminated.
2. This clause overrides any existing configuration options for automatic skipping and for the automatic termination of an ACCEPT statement. Refer to the *User's Guide* for details of configuration options.
3. In a fixed-format numeric-edited screen item, the AUTO clause causes the decimal point position to be skipped automatically once all of the integer places have been filled. Selection of fixed format mode is a configuration option. Refer to the *User's Guide* for details of configuration options.

BACKGROUND-COLOR Clause

Function

The BACKGROUND-COLOR clause specifies the background color of the screen item.

General Format

The following figure shows the format of the BACKGROUND-COLOR clause:



Syntax Rules

The following syntax rules apply to the BACKGROUND-COLOR clause:

1. BACKGROUND-COLOR and BACKGROUND-COLOUR are synonymous.
2. This clause is allowed with any screen item.
3. If this clause is specified at group level, it applies to all elementary subordinate items.
4. integer-1 must be a value from 0 to 7.

General Rules

The following general rules apply to the BACKGROUND-COLOR clause:

1. This clause is only available for use with a color screen.
2. integer-1 specifies the background color of the screen item. The colors and their corresponding values are:

0	black
1	blue
2	green
3	cyan
4	red
5	magenta
6	brown
7	white
3. If this clause is not specified, the background color defaults to black.
4. If a screen description contains a BLANK SCREEN clause, and either contains a BACKGROUND-COLOR clause or is subordinate to one that does, then when the screen item is displayed by a DISPLAY statement the specified color becomes the default background color. It remains the default background color until either another screen item with this combination of options is displayed (whether in the same DISPLAY statement or another), or a Format 2 DISPLAY statement with both options is executed.

-
5. Upon initial execution of your program, the AIX VS COBOL system assumes that the current colors displayed on the screen are white for the foreground and black for the background. Therefore, if the first DISPLAY statement tries to set the foreground color to white or the background color to black, then the AIX VS COBOL system leaves the color alone since it thinks the color is already set correctly. If your program is affected by this you may do one of two things:
 - a. Set the terminal colors to white (foreground) and black (background) using the AIX system **chcolor** command before entering the COBOL application.
 - b. DISPLAY colors other than foreground white or background black as your first DISPLAY statement in your COBOL program.

BELL Clause

Function

The BELL clause causes an audible alarm to sound each time the item containing the clause is accepted or displayed.

General Format

The following figure shows the format of the BELL clause:



Syntax Rules

The following syntax rules apply to the BELL clause:

1. BELL and BEEP are synonymous.
2. This clause is only allowed with elementary items.

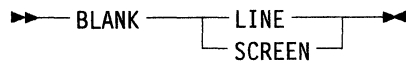
BLANK Clause

Function

The BLANK clause is effective each time the screen item containing the clause is displayed. BLANK LINE erases from the current cursor position to the end of the current line. BLANK SCREEN erases the entire display screen and places the cursor at line 1, column 1.

General Format

The following figure shows the format of the BLANK clause:



Syntax Rule

The BLANK clause is allowed only with elementary items.

General Rules

The following general rules apply to the BLANK clause:

1. This clause has no effect in an ACCEPT operation.
2. The erasing is done before the item is displayed.
3. The BLANK SCREEN clause causes the display screen to return to its default foreground and background colors. For additional effects, if the screen item is subject to a FOREGROUND-COLOR or BACKGROUND-COLOR clause, refer to the sections on those clauses.

Example

```
01  PASSWORD-SCREEN.  
02  BLANK SCREEN.  
02  LINE 5 COLUMN 5 BLANK LINE  
    VALUE "ENTER PASSWORD".
```

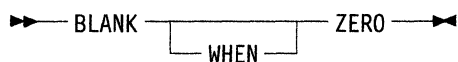
BLANK WHEN ZERO Clause

Function

The BLANK WHEN ZERO clause causes the blanking of a screen item when its value is zero. Refer to Chapter 6, "Data Division" for standard rules for this data description clause which also apply when the BLANK WHEN ZERO clause is used in screen definitions.

General Format

The following figure shows the format of the BLANK WHEN ZERO clause:



Syntax Rule

The following rule applies to the BLANK WHEN ZERO clause:

1. This clause is allowed only with input, output, and update fields that are numeric or numeric-edited.

BLINK Clause

Function

The BLINK clause causes the screen item to blink when it appears on the screen.

General Format

The following figure shows the format of the BLINK clause:

▶— BLINK —▶

Syntax Rules

The following syntax rules apply to the BLINK clause:

1. The BLINK clause is allowed with any screen item.
2. If the BLINK clause is specified at group level, it applies to all suitable subordinate elementary items.
3. The BLINK clause is always accepted syntactically. However, it will only have effect at run time if the display being used supports blinking.

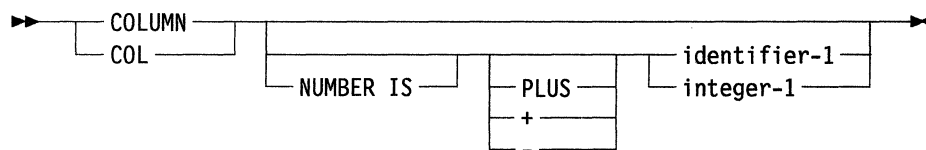
COLUMN Clause

Function

The COLUMN clause specifies the column at which the screen item starts on the screen.

General Format

The following figure shows the format of the COLUMN clause:



Syntax Rules

The following syntax rules apply to the COLUMN clause:

1. identifier-1 must be an unsigned numeric integer and must contain a value greater than zero and less than 256.
2. identifier-1 must not be subject to OCCURS clauses.
3. integer-1 must be unsigned, greater than zero, and less than 256.
4. PLUS and + are synonymous.
5. COL is an abbreviation for COLUMN.
6. Omitting the NUMBER option results in a default value of +1.
7. COLUMN 1 is assumed for screen descriptions that specify the LINE clause but omit the COLUMN clause.
8. The COLUMN clause may be specified with any elementary item.

General Rules

The following general rules apply to the COLUMN clause:

1. The COLUMN clause specifies the column in which the screen item is to appear on the display screen in an ACCEPT or DISPLAY operation.
2. If the COLUMN clause has an identifier or an integer but does not specify PLUS, +, or -, the clause gives an absolute column number. Column 1 is the column number specified in the AT phrase of the statement. If the AT phrase is not specified, then column 1 is the first column on the screen.

-
3. If PLUS, +, or - is specified in the COLUMN clause, then the column number is relative to that at which the preceding screen item ends, regardless of whether the statement displays the preceding item on the screen. This depends on the current effective length of that item at run time, derived from its PICTURE, VALUE, and SIZE clauses. The counting of column numbers restarts at a level 01 item at column 1.
 4. If a screen description contains neither a LINE or COLUMN clause, and the item is not a level 01 item, COLUMN + 1 is assumed. The item then starts immediately following the preceding item in the SCREEN SECTION.
 5. If a COLUMN position is specified that is off the screen, wraparound occurs to the next (or previous) line.

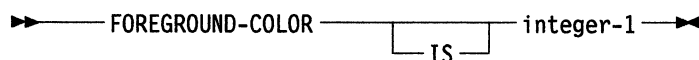
FOREGROUND-COLOR Clause

Function

The FOREGROUND-COLOR clause specifies the foreground color of the screen item.

General Format

The following figure shows the format of the FOREGROUND-COLOR clause:



Syntax Rules

The following syntax rules apply to the FOREGROUND-COLOR clause:

1. FOREGROUND-COLOR and FOREGROUND-COLOUR are synonymous.
2. This clause is allowed with any screen item.
3. If this clause is specified at group level, it applies to all elementary subordinate items.
4. integer-1 must be a value from 0 through 7.

General Rules

The following general rules apply to the FOREGROUND-COLOR clause:

1. This clause is only available for use with a color screen.
2. integer-1 specifies the foreground color of the screen item. The colors and their corresponding values are:

0	black
1	blue
2	green
3	cyan
4	red
5	magenta
6	brown
7	white
3. If this clause is not specified, the foreground color defaults to white.
4. If a screen description contains a BLANK SCREEN clause, and either contains a BACKGROUND-COLOR clause or is subordinate to one that does, then when the screen item is displayed by a DISPLAY statement the specified color becomes the default foreground color. It remains the default foreground color until either another screen item with this combination of options is displayed (whether in the same DISPLAY statement or another), or a Format 2 DISPLAY statement with both options is executed.

-
5. If the **HIGHLIGHT** clause is also specified, foreground colors are brightened.
 6. Upon initial execution of your program, the AIX VS COBOL system assumes that the current colors displayed on the screen are white for the foreground and black for the background. Therefore, if the first **DISPLAY** statement tries to set the foreground color to white or the background color to black, then the AIX VS COBOL system leaves the color alone since it thinks the color is already set correctly. If your program is affected by this you may do one of two things:
 - a. Set the terminal colors to white (foreground) and black (background) using the AIX system **chcolor** command before entering the COBOL application.
 - b. **DISPLAY** colors other than foreground white or background black as your first **DISPLAY** statement in your COBOL program.

Example

The following example shows the **BACKGROUND-COLOR** clause:

```
05  LINE 2  PIC X(10)  TO EMP-NAME BACKGROUND-COLOR IS 1
                                FOREGROUND-COLOR IS 4.
```

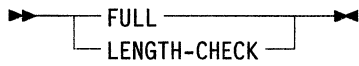
FULL Clause

Function

The FULL clause specifies that the operator must either leave the screen item completely empty or fill it entirely with data.

General Format

The following figure shows the format of the FULL clause:



Syntax Rules

The following syntax rules apply to the FULL clause:

1. FULL and LENGTH-CHECK are synonymous.
2. The FULL clause is valid only in input and update fields and group items.
3. If this clause is specified at group level, it applies to all suitable subordinate elementary items.

General Rules

The following general rules apply to the FULL clause:

1. The FULL clause is effective during the execution of any ACCEPT statement that causes the screen item to be accepted provided the cursor enters the screen item at some time during the ACCEPT operation. Until this clause is satisfied, terminator keystrokes are rejected, and the cursor is repositioned to the end of the item. If the item is fixed-format numeric-edited, the cursor is repositioned to the decimal point position.
2. If the screen item is alphanumeric, then to satisfy this clause, either the entire item must contain only spaces or prompt characters, or both the first and last character positions must contain non-space, nonprompt characters.
3. If the screen item is free-format numeric or free-format numeric-edited, then to satisfy the clause, either the resultant value must be zero, or both the first and last character positions must contain non-space, nonprompt characters.
4. If the screen item is fixed-format numeric-edited, then to satisfy the clause either the value must be zero or there must be no digit positions in which zero-suppression has taken effect.
5. The FULL clause has no effect on fixed-format numeric or on numeric-edited screen items that have no zero-suppression positions.
6. For update fields, the FULL clause can be satisfied by initial data as well as operator-keyed data.

-
7. The FULL clause may not be effective if a function key is used to terminate the ACCEPT operation. Refer to the *User's Guide* for details of configuration options.
 8. An error message may be configured for display on the display screen if the FULL clause is not satisfied. Refer to the *User's Guide* for details of configuration options.

GRID Clause

Function

The GRID clause causes each character of the screen item to have a vertical line on its left-hand side when the item appears on the screen. Each line is within the character-position.

General Format

The following figure shows the format of the GRID clause:

►►— GRID —◄◄

Syntax Rules

The following syntax rules apply to the GRID clause:

1. The GRID clause can be used with any screen item.
2. If the GRID clause is specified at group level, it applies to all suitable subordinate elementary items.
3. This clause is provided for compatibility. It is accepted syntactically but has no effect at run time.

HIGHLIGHT Clause

Function

The HIGHLIGHT clause causes the screen item to appear in high-intensity mode when it appears on the screen.

General Format

The following figure shows the format of the HIGHLIGHT clause:

▶— HIGHLIGHT —▶

Syntax Rules

The following syntax rules apply to the HIGHLIGHT clause:

1. This clause is valid for input, output, update, or literal fields.
2. If the HIGHLIGHT clause is specified at group level, it applies to all suitable subordinate elementary items.

General Rule

If the FOREGROUND-COLOR clause is also specified, the HIGHLIGHT clause causes the foreground colors to become brighter.

Example

The following example shows the HIGHLIGHT clause:

```
01 PERSON-INFO.  
02 BLANK SCREEN.  
02 LINE 5 COLUMN 5 PIC X(20) FROM EMP-NAME HIGHLIGHT.  
02 LINE 7 COLUMN 5 PIC X(40) FROM EMP-ADDR.
```

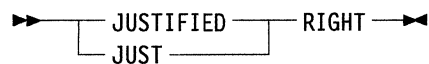
JUSTIFIED Clause

Function

The JUSTIFIED clause specifies nonstandard positioning of data within a screen item when data is either moved to it or entered into it. Refer to Chapter 6, “Data Division” for standard rules for this data description clause which are also applicable when the JUSTIFIED clause is used in screen definitions.

General Format

The following figure shows the format for the JUSTIFIED clause:



Syntax Rules

The following syntax rules apply to the JUSTIFIED clause:

1. JUST is an abbreviation for JUSTIFIED.
2. The JUSTIFIED clause is allowed only with input, output, and update fields.

General Rules

The following general rules apply to the JUSTIFIED clause:

1. If the screen item is an output or update field, when data is moved to it from the sending item, the JUSTIFIED clause is applied according to the normal rules for MOVE. Refer to Chapter 3, “Introduction to the Nucleus” on page 3-1.
2. If the screen item is an input or update field, when the accepting of data into it has terminated, the JUSTIFIED clause causes the data entered to be moved right by the number of character positions occupied by prompt characters, and the left of the field to be padded with spaces. This is done before the data is moved to the receiving item. This does not occur if data is not entered.
3. If the screen item has a SECURE clause, the effect on the data is the same as it would be without the presence of the SECURE clause. This effect does not appear on the screen.

LEFTLINE Clause

Function

The LEFTLINE clause causes the leftmost character of the screen item to have a vertical line on its left-hand side when the item appears on the screen. The line is within the character-position.

General Format

The following figure shows the format of the LEFTLINE clause:

▶— LEFTLINE —▶

Syntax Rules

The following syntax rules apply to the LEFTLINE clause:

1. The LEFTLINE clause can be used with any screen item.
2. If the LEFTLINE clause is specified at group level, it applies to all suitable subordinate elementary items.
3. This clause is provided for compatibility. It is accepted syntactically but has no effect at run time.

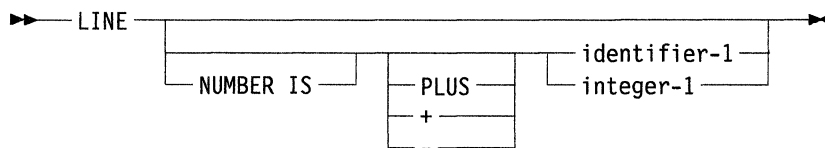
LINE Clause

Function

The **LINE** clause specifies the line at which the screen item starts on the screen.

General Format

The following figure shows the format of the **LINE** clause:



Syntax Rules

The following syntax rules apply to the **LINE** clause:

1. identifier-1 must be an unsigned numeric integer and must contain a value greater than zero and less than 256.
2. identifier-1 must not be subject to **OCCURS** clauses.
3. integer-1 must be unsigned, greater than zero, and less than 256.
4. **PLUS** and **+** are synonymous.
5. Omitting the **NUMBER** option results in a default value of **+1**.
6. The current line is assumed for screen descriptions which omit the **LINE** clause.
7. The **LINE** clause may be specified with any elementary item.

General Rules

The following general rules apply to the **LINE** clause:

1. The **LINE** clause specifies the line on which the screen item is to appear on the display screen in an **ACCEPT** or **DISPLAY** operation.
2. If the **LINE** clause has an identifier or an integer but does not specify **PLUS**, **+**, or **-**, the clause gives an absolute line number. Line 1 is the line number specified in the **AT** phrase of the statement. If the **AT** phrase is not specified, then line 1 is the first line on the screen.
3. If **PLUS**, **+**, or **-** is specified in the **LINE** clause, then the line number is relative to that at which the preceding display screen item ends, regardless of whether the statement displays the preceding item on the screen. This depends on the current effective length of that item at run time, derived from its **PICTURE**, **VALUE**, and **SIZE** clauses. The counting of line numbers restarts at a level 01 item at line 1.

-
4. If a screen description contains neither a `LINE` nor `COLUMN` clause and the item is not a level 01 item, `COLUMN +1` is assumed. The item then starts immediately following the preceding item in the screen section.
 5. If a `LINE` position is specified that is off the screen, the `ACCEPT` or `DISPLAY` is truncated.

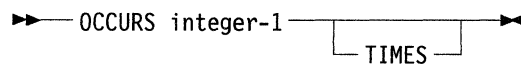
OCCURS Clause

Function

The OCCURS clause eliminates the need for separate entries for repeated screen items and supplies information required for the application of subscripts or indexes. Refer to Chapter 12, "Table-Handling" for standard rules for this data description clause which are also applicable when the OCCURS clause is used in screen definitions.

General Format

The following figure shows the format of the OCCURS clause:



Syntax Rules

The following syntax rules apply to the OCCURS clause:

1. This clause is allowed only with input, output, update fields, and with group items.
2. If OCCURS clauses apply to the screen item, then the same number of OCCURS clauses, specifying the same number of occurrences, must apply to the receiving item. These OCCURS clauses must not include the DEPENDING phrase.
3. If OCCURS clauses apply to the screen item, either the same number of OCCURS clauses, specifying the same number of occurrences or no OCCURS clauses at all, must apply to the sending item. These OCCURS clauses must not include the DEPENDING phrase.

General Rules

The following general rules apply to the OCCURS clause:

1. In a screen description that is subject to an OCCURS clause, the LINE and COLUMN clauses apply to each individual table entry. Either the LINE or the COLUMN clause should specify relative positioning, because if they both specify an absolute position every entry will appear in the same place.
2. If the screen item is an output field and no OCCURS clauses apply to the sending item, then in a DISPLAY operation the contents of the sending item are moved to every occurrence of the screen item.
3. If the screen item is an update field or it is an output field with OCCURS clauses applying to the sending item, then in a DISPLAY operation the contents of each occurrence of the sending item are moved to the corresponding occurrence of the screen item.
4. If the screen item is an update or input field then in an ACCEPT operation the data entered into each occurrence of the screen item is moved to the corresponding occurrence of the receiving item.

Example

The following example shows the OCCURS clause:

```
WORKING-STORAGE SECTION.  
01 NAMES.  
   05 NAME-TABLE PIC X(20) OCCURS 10 TIMES.  
   :  
   :  
SCREEN SECTION.  
01 STUDENT-NAMES.  
   05 BLANK SCREEN.  
   05 LINE 5 VALUE "ENTER STUDENT NAMES:".  
   05 LINE 8.  
   05 LINE +2 COLUMN 2 PIC X(20)  
      TO NAME-TABLE OCCURS 10.
```

OVERLINE Clause

Function

The **OVERLINE** clause causes every character of the screen item to have a horizontal line above it when the item appears on the screen. The line is within the character position.

General Format

The following figure shows the format of the **OVERLINE** clause:

▶— OVERLINE —▶

Syntax Rules

The following syntax rules apply to the **OVERLINE** clause:

1. The **OVERLINE** clause can be used with a screen item.
2. If the **OVERLINE** clause is specified at group level, it applies to all suitable subordinate elementary items.
3. This clause is provided for compatibility. It is accepted syntactically but has no effect at run time.

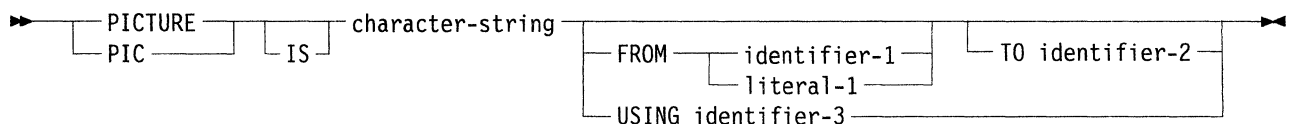
PICTURE Clause

Function

The PICTURE clause describes the length, general characteristics, and editing requirements of a screen item. The FROM, TO, and USING phrases identify the source of data for display and the destination of data accepted.

General Format

The following figure shows the format of the PICTURE clause:



Syntax Rules

The following syntax rules apply to the PICTURE clause:

1. The PICTURE clause may include any standard editing characters.
2. Each PICTURE clause must contain either a FROM, TO, or USING phrase.
3. PIC is an abbreviation for PICTURE.
4. The USING phrase is equivalent to the TO and FROM phrases, both specifying the same identifier.
5. The PICTURE clause is allowed only with elementary items.
6. The identifier in the FROM, TO, or USING phrase can be qualified. If no OCCURS clause applies to the screen item, then this identifier can be subscripted or indexed. It must be defined in the FILE, WORKING-STORAGE, or LINKAGE section of the program.
7. The PICTURE clause need not be the same as the PICTURE clause of the data item referenced in the FROM, TO, or USING clause, but it must be such that the implied MOVE is legal.

General Rules

The following general rules apply to the PICTURE clause:

1. The character string describes the length and category of the screen item. It is used in the same way as the character string in the PICTURE clause for a data item.
2. Executing a DISPLAY statement on a screen item whose description includes a FROM or USING phrase moves data from the associated data item to the screen item and then displays the screen item on the screen.

-
3. Executing an ACCEPT statement on a screen item whose description includes a TO or USING phrase accepts operator-keyed data into the screen item and then moves that data to the associated data item.
 4. It is recommended that every numeric screen item either be a numeric-edited item or contain only 9s in its PICTURE clause. Editing and de-editing are applied as necessary when data is moved to or from the associated data item.

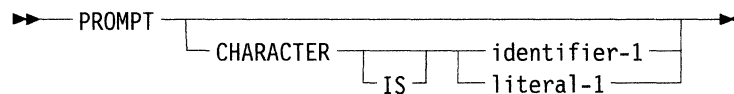
PROMPT Clause

Function

The PROMPT clause causes the empty character positions in the screen item to be marked on the screen during an ACCEPT operation while the system is ready to accept operator-keyed data into that item.

General Format

The following figure shows the format of the PROMPT clause:



Syntax Rules

The following syntax rules apply to the PROMPT clause:

1. This clause is allowed only with input and update fields and with group items.
2. If this clause is specified at group level, it applies to all subordinate elementary items.
3. identifier-1 must be a single-character alphabetic or alphanumeric data item.
4. identifier-1 must not be subject to an OCCURS clause.
5. literal-1 must be a one-character nonnumeric literal or a figurative constant.

General Rules

The following general rules apply to the PROMPT clause:

1. If the CHARACTER phrase is not specified, the PROMPT clause is documentary only.
2. The CHARACTER phrase specifies a *prompt character* to be used for marking empty character positions. The prompt character overrides the configured option. Refer to the *User's Guide* for details on configuration options.
3. The PROMPT clause causes the prompt character to replace trailing spaces in alphanumeric or free-format numeric screen items. It also causes the prompt character to replace leading suppressed digit positions in fixed-format numeric-edited screen items.
4. The PROMPT clause has no effect on fixed-format, nonedited numeric screen items or numeric-edited screen items that have no zero-suppression positions.
5. This clause has no effect if the SECURE clause is specified.
6. The prompt characters appearing in the screen item are changed to spaces upon termination of the ACCEPT operation.
7. Using the left or right arrow key to try to move over a prompted area terminates the ACCEPT into the screen item, unless there is no further item in the applicable direction.

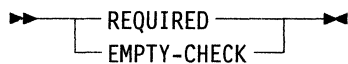
REQUIRED Clause

Function

The REQUIRED clause specifies that the operator must not leave the screen item empty.

General Format

The following figure shows the format of the REQUIRED clause:



Syntax Rules

The following syntax rules apply to the REQUIRED clause:

1. REQUIRED and EMPTY-CHECK are synonymous.
2. The REQUIRED clause is allowed only with input and update fields and with group items.
3. This clause may be specified on a group screen item, in which case it applies to all suitable elementary items which are subordinate to that item.

General Rules

The following general rules apply to the REQUIRED clause:

1. The REQUIRED clause takes effect during the execution of any ACCEPT statement that causes the screen item to be accepted as long as the cursor enters the screen item at some time during the ACCEPT statement. Unless this clause is satisfied, terminator keystrokes are rejected and the cursor is repositioned to the beginning of the item.
2. To satisfy this clause, alphanumeric screen items must contain at least one nonspace, nonprompt character; numeric screen items must have a nonzero value.
3. For update fields, the REQUIRED clause can be satisfied by initial data as well as by operator-keyed data.
4. The REQUIRED clause may not be effective if a function key is used to terminate an ACCEPT operation. Refer to the *User's Guide* for details on configuration options.
5. An error message may be configured for display on the screen if the REQUIRED clause is not satisfied. Refer to the *User's Guide* for details on configuration options.

REVERSE-VIDEO Clause

Function

The REVERSE-VIDEO clause causes the screen item to be displayed in reverse-video.

General Format

The following figure shows the format of the REVERSE-VIDEO clause:

►► REVERSE-VIDEO ◄◄

General Rules

The following general rules apply to the REVERSE-VIDEO clause:

1. The REVERSE-VIDEO clause is allowed with any screen item.
2. If the REVERSE-VIDEO clause is specified at group level, it applies to all suitable subordinate elementary items.

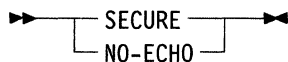
SECURE Clause

Function

The SECURE clause prevents operator-keyed data from appearing on the screen.

General Format

The following figure shows the format of the SECURE clause:



Syntax Rules

The following syntax rules apply to the SECURE clause:

1. The SECURE clause is allowed only with input and update fields.
2. SECURE and NO-ECHO are synonymous.
3. This clause may be specified in a group screen item, in which case it applies to all suitable elementary items which are subordinate to that item.

General Rule

When the SECURE clause is specified, only spaces and the cursor appear in the screen item.

Example

The following example shows the SECURE clause:

```

01  PASSWORD-SCREEN.
    02  BLANK SCREEN.
    02  LINE 5 VALUE "ENTER PASSWORD".
    02  LINE 6 PIC X(6) TO PASS-WD SECURE.

```

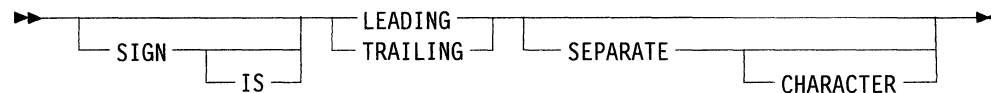
SIGN Clause

Function

The SIGN clause specifies the position and representation of the operational sign. Refer to Chapter 6, "Data Division" for standard rules for this data description clause which are also applicable when the SIGN clause is used in screen definitions.

General Format

The following figure shows the format of the SIGN clause:



Syntax Rules

The following syntax rules apply to the SIGN clause:

1. The SIGN clause is allowed only with input, output, and update fields whose pictures contain the character S.
2. This clause is allowed only with elementary items.

General Rule

It is recommended that the SEPARATE option be used when the SIGN clause is specified in a screen description. If the SEPARATE option is not specified, a sign denoted by S in a PICTURE clause appears as an overwrite.

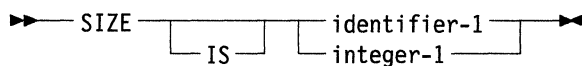
SIZE Clause

Function

The SIZE clause specifies the current size of the screen item.

General Format

The following figure shows the format of the SIZE clause:



Syntax Rules

The following syntax rules apply to the SIZE clause:

1. The SIZE clause is allowed only with elementary screen items.
2. identifier-1 must be an unsigned numeric integer and must not be subject to an OCCURS clause.
3. integer-1 must be unsigned.

General Rules

The following general rules apply to the SIZE clause:

1. The SIZE clause has no effect if the size specified is zero.
2. If the SIZE clause is specified for a numeric or numeric-edited screen item and the size specified is not zero, the screen item is treated as though it were free-format. This overrides the setting of the configuration option.
3. If the size specified in the SIZE clause is less than that implied by the associated PICTURE or VALUE clause, only the left-hand portion of the screen item appears on the screen. If the JUSTIFIED clause is present, then only the right-hand portion of the screen item appears. The remainder of the screen item can be considered to contain spaces or zeros, as appropriate.
4. If the size specified in the SIZE clause is greater than that implied by the PICTURE or VALUE clause for output or literal fields, the screen item is padded on the right with spaces.
5. Changing the value in identifier-1 alters the effective size of the screen item at run time. This may alter the screen positions of items whose descriptions follow it in the SCREEN SECTION. Refer to "LINE Clause" on page 18-38 and "COLUMN Clause" on page 18-28.

UNDERLINE Clause

Function

The UNDERLINE clause causes the screen item to be underlined when it appears on the screen.

General Format

The following figure shows the format of the UNDERLINE clause:

►►— UNDERLINE —◄◄

Syntax Rules

The following syntax rules apply to the UNDERLINE clause:

1. The UNDERLINE clause is allowed with any screen item.
2. If the UNDERLINE clause is specified at group level, it applies to all suitable subordinate elementary items.

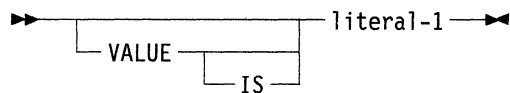
VALUE Clause

Function

The VALUE clause specifies literal information for display on the screen.

General Format

The following figure shows the format of the VALUE clause:



Syntax Rules

The following syntax rules apply to the VALUE clause:

1. The literal associated with the VALUE clause must be nonnumeric. It cannot be a figurative constant.
2. The VALUE clause is allowed only with elementary items that have no PICTURE clauses.

ZERO-FILL Clause

Function

The ZERO-FILL clause causes trailing prompt characters to be replaced by zeros instead of spaces.

General Format

The following figure shows the format of the ZERO-FILL clause:

►— ZERO-FILL —►

Syntax Rule

This clause is allowed only with input and update fields that are alphabetic or alphanumeric.

General Rules

The following general rules apply to the ZERO-FILL clause:

1. The ZERO-FILL clause causes trailing prompt characters to be replaced by zeros instead of spaces when data is moved from the screen item to the receiving item. This occurs only if the operator enters data into the screen item.
2. If the receiving item has a JUSTIFIED clause, the ZERO-FILL clause causes leading positions left vacant by justification to be zero-filled.

Procedure Division

The formats of the ACCEPT and DISPLAY statements described in this section allow non-scrolling forms to be accepted or displayed. The operator can then enter data into a displayed form.

The order of execution of an ACCEPT or DISPLAY statement is always:

1. The AT phrase
2. The BLANK phrase
3. Display of current contents of the data area (ACCEPT only)
4. The BELL phrase
5. The ACCEPT or DISPLAY operation.

Format 2 of both ACCEPT and DISPLAY statements contains a WITH phrase that allows the user to specify certain options available during the operation. Many of these options are also available as screen description clauses and have been documented earlier in this chapter. Refer to "SCREEN SECTION" on page 18-14 for descriptions of:

AUTO
BACKGROUND-COLOR
BELL
BLINK
FOREGROUND-COLOR
FULL
GRID
HIGHLIGHT
LEFTLINE
OVERLINE
PROMPT
REQUIRED
REVERSE-VIDEO
SECURE
SIZE
UNDERLINE
ZERO-FILL (with alphabetic or alphanumeric data items)

In addition to the options available as screen description clauses, the following options can be used in the WITH phrase: SPACE-FILL, ZERO-FILL, LEFT-JUSTIFY, RIGHT-JUSTIFY, TRAILING-SIGN, and UPDATE. ZERO-FILL appears in this list and as a screen description clause because it has two different uses. Its second use is documented later in this chapter.

A configuration option is available that allows the entry of data into numeric and numeric-edited screen fields in free-format mode. In COBOL, nonedited numeric data items are intended for holding data in an internal form; however, Format 2 ACCEPT and DISPLAY statements enable such data items to appear on the screen. If free-format mode is in effect, the data will automatically appear reformatted as follows with:

- The virtual decimal point represented by a period
- The sign represented by a sign character (- for minus; space for plus) which appears immediately before the leftmost digit
- Zero suppression in all integer character positions, except the least significant
- Left justification

The SPACE-FILL, ZERO-FILL, LEFT-JUSTIFY, RIGHT-JUSTIFY, and TRAILING-SIGN options amend this format.

All of the additional options available with the WITH phrase are described under the rules for ACCEPT and/or DISPLAY formats.

ACCEPT Statement

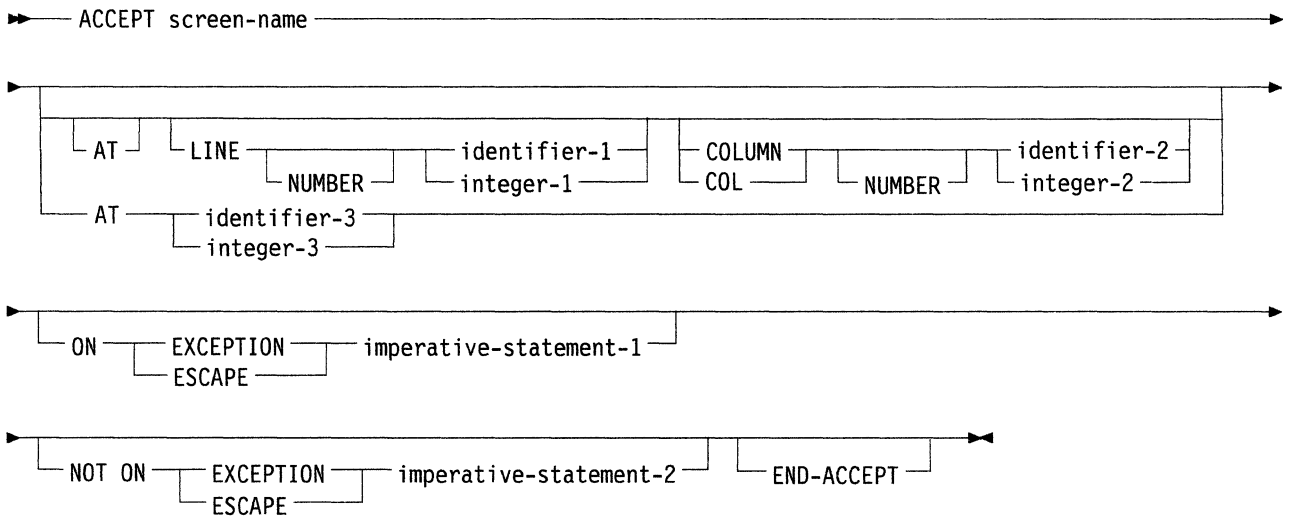
Function

The ACCEPT statement makes data keyed at the CRT available to your program. Refer to Chapter 7, "Procedure Division in the Nucleus" for other formats of the ACCEPT statement.

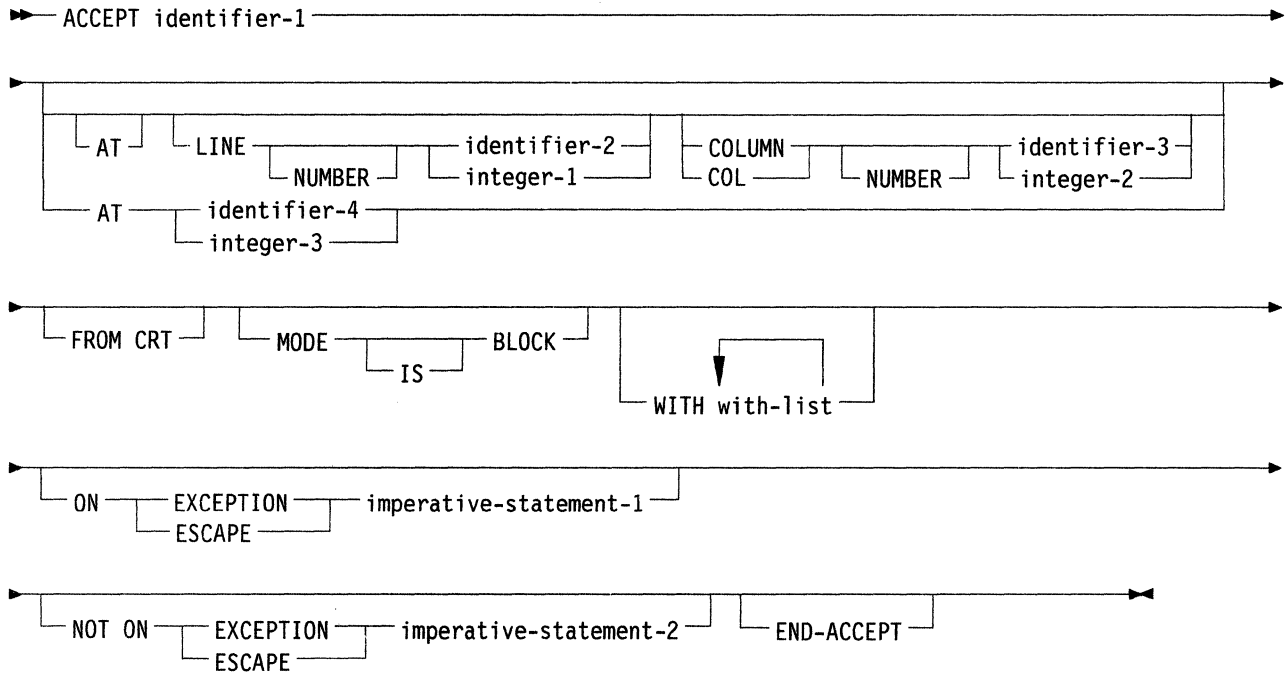
General Format

The following figures show the format of the ACCEPT statement:

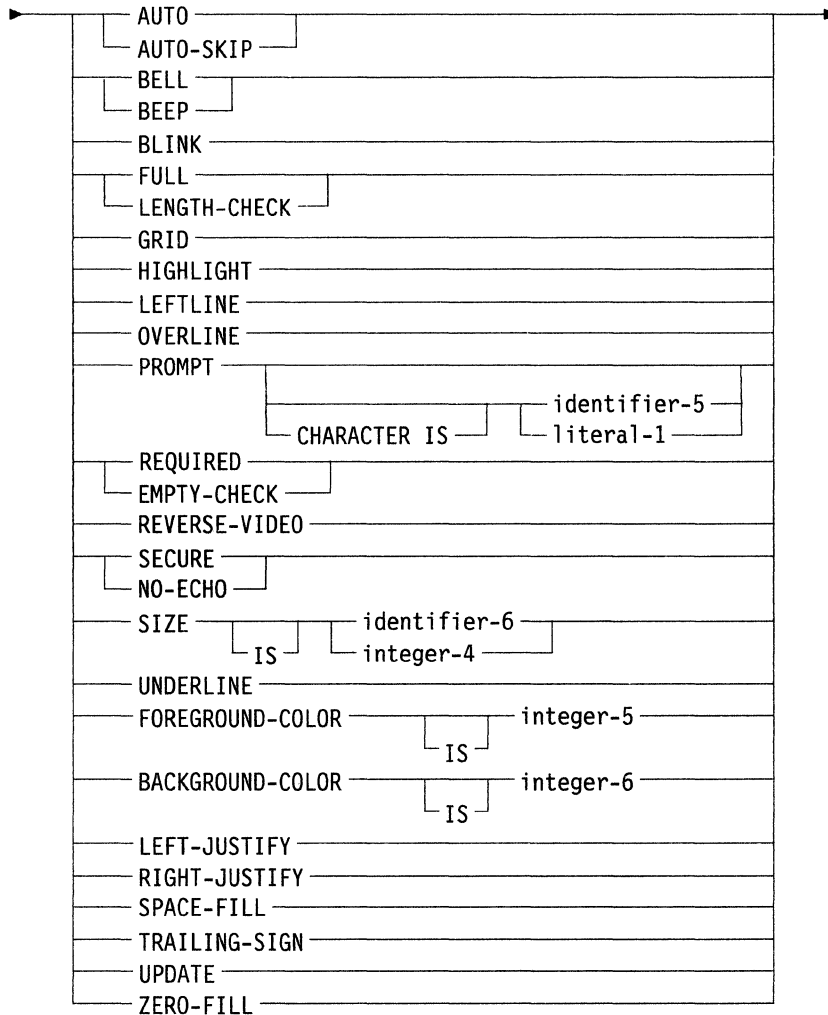
Format 1



Format 2



where with-list is:



Syntax Rules

The following syntax rules apply to the ACCEPT statement:

Both Formats

1. The LINE and COLUMN phrases can appear in any order.
2. EXCEPTION and ESCAPE are synonymous.
3. integer-3 must be 4 or 6 digits long.

Format 1

4. identifier-3 must be a PIC 9(4) or a PIC 9(6) data item.

Format 2

5. identifier-4 must be a PIC 9(4) or a PIC 9(6) data item.

-
6. An ACCEPT statement whose operand is an identifier is treated as Format 2 if it has:
 - a. An AT phrase
 - b. A FROM phrase with the CRT option
 - c. A WITH phrase
 - d. A MODE IS BLOCK phrase
 - e. An EXCEPTION phrase
 - f. No FROM phrase but the CONSOLE option clause is specified in the SPECIAL-NAMES paragraph.

If it has the FROM phrase with the CONSOLE option, or if it has no FROM phrase and the CONSOLE IS CRT clause is not specified in the SPECIAL-NAMES paragraph, it is treated as the standard ANSI ACCEPT statement. Refer to "ACCEPT Statement" on page 7-22.

7. The phrases following the identifier can be in any order.
8. If the PROMPT option is specified, the CHARACTER option must specify a literal and not an identifier.
9. The SPACE-FILL, ZERO-FILL, LEFT-JUSTIFY, RIGHT-JUSTIFY and TRAILING-SIGN options are allowed only if the operand is an elementary item.
10. The attributes GRID, LEFTLINE, and OVERLINE are provided for compatibility. They are accepted syntactically but have no effect at run time.
11. The attribute BLINK is always accepted syntactically but will have effect only on a terminal that supports blinking.

General Rules

The following general rules apply to the ACCEPT statement:

Both Formats

1. The AT phrase gives the absolute address on the screen where the ACCEPT operation is to start.
2. If integer-3 is 4 digits long, the first two digits specify the line, the second two the column. If integer-3 is 6 digits long, the first three digits specify the line, while the second three specify the column.
3. Certain combinations of line and column numbers have special meanings:
 - a. Until the column comes within range, out-of-range column values are reduced by the line length and the line value is incremented.
 - b. Out-of-range line values cause the screen to scroll up one line. The effect is the same as if the line number of the bottom line had been specified.
 - c. If the line and column numbers given are both zero, the ACCEPT starts at the position following the position where the preceding Format 1 or Format 2 ACCEPT operation finished. Column 1 of each line is considered to follow the last column of the previous line.
 - d. If the line number is zero, but a nonzero column number is specified, the ACCEPT starts at the specified column, on the line following the line where the preceding Format 1 or 2 ACCEPT operation finished.
 - e. If the column number is zero, but a nonzero line number is specified, the ACCEPT starts on the specified line, at the column following the column where the preceding Format 1 or 2 ACCEPT operation finished.

-
4. If the ON EXCEPTION phrase is specified, imperative-statement-1 is executed if the ACCEPT operation finishes with anything other than a normal termination. If the NOT ON EXCEPTION phrase is specified, imperative-statement-2 is executed if the ACCEPT operation terminates normally. Refer to "CRT STATUS Clause" on page 18-12 for possible types of termination.
 5. The END-ACCEPT phrase delimits the scope of the ACCEPT statement. Refer to "Explicit and Implicit Scope Terminators" on page 2-36.

Format 1

6. If identifier-3 is 4 digits long, the first two digits specify the line, the second two the column. If identifier-3 is 6 digits long, the first three digits specify the line, the second three the column.
7. This format of the ACCEPT statement accepts screen items which are defined within the SCREEN SECTION of the program and allows full access to the enhanced screen-handling facilities.

Format 2

8. If identifier-4 is 4 digits long, the first two digits specify the line, the second two the column. If identifier-4 is 6 digits long, the first three digits specify the line, the second three the column.
9. If no AT phrase is specified, the ACCEPT operation starts at line 1, column 1.
10. If identifier is a group item and there is no MODE IS BLOCK phrase, then those elementary subordinate items which have names other than FILLER are accepted. They are positioned on the screen in the order their descriptions appear in the DATA DIVISION and are separated by the lengths of the FILLER items in the group. For this purpose, the first position on a line is regarded as immediately following the last position on the previous line. The items are accepted in the same order. Unless otherwise specified in the CURSOR IS clause, the cursor is initially positioned at the start of the first item. Refer to "CURSOR IS Clause" on page 18-11. As the ACCEPT operation into each item is terminated, the cursor moves to the start of the next item.
11. The MODE IS BLOCK phrase indicates that the identifier is to be treated as an elementary item. Even if it is a group item it is displayed as one item.
12. If the PROMPT option is not specified, no character is output to mark empty character positions. PROMPT without the CHARACTER option causes this to be done using the character configured for this purpose.
13. If the PROMPT option is not specified, those character positions into which the operator does not enter data produce spaces in the data item.
14. The SPACE-FILL option causes data in free-format, nonedited numeric data items to appear on the screen with zero-suppression in all integer character positions. This option affects only free-format, nonedited numeric data items. This takes effect when initial data in the data item is displayed and again when the ACCEPT operation into the data item is terminated. Any leading sign is displayed in the rightmost space.
15. The ZERO-FILL option causes data in free-format, nonedited numeric data items to appear on the screen with no zero-suppression. This takes effect when initial data in the data item is displayed and again when the ACCEPT operation into the data item is terminated. Refer to "ZERO-FILL Clause" on page 18-53 for the effect this option has when used with alphabetic or alphanumeric data items.
16. The LEFT-JUSTIFY option is documentary only.
17. The RIGHT-JUSTIFY option causes operator-keyed characters to be moved on the screen to the rightmost character positions of the field. This option affects only free-format, nonedited numeric data items. This takes effect upon display of the initial data (the current contents displayed) in the data item and also upon termination of the ACCEPT operation.

-
18. The TRAILING-SIGN option causes the operational sign to appear in the rightmost character position of the field. This takes effect upon display of initial data in the data item and also upon termination of the ACCEPT operation. This option affects only signed, nonedited numeric data items which are in free-format mode.
 19. The UPDATE option causes the current contents (initial data) of the data item to be displayed before the operator is prompted to key in new input. If the operator does not key in any new data, the initial data is then treated as though it were operator-keyed. If the UPDATE option is not specified, the display of initial data is a configuration option. Refer to the *User's Guide* for details of configuration options.

DISPLAY Statement

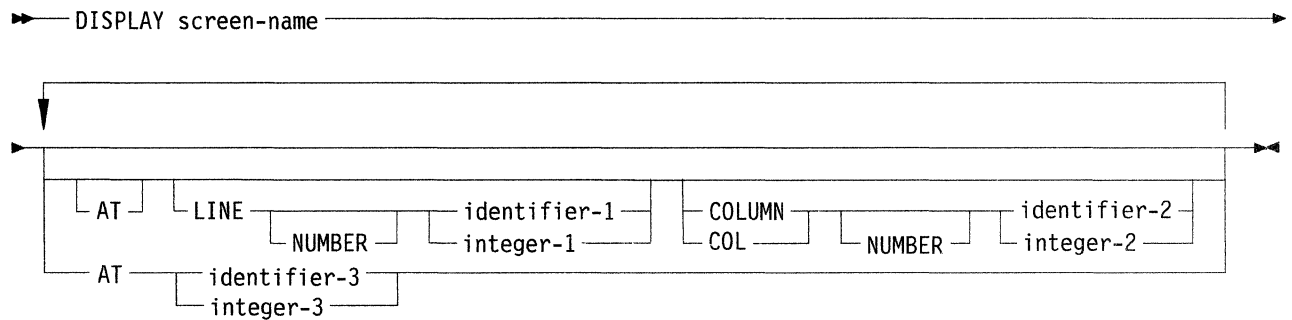
Function

The DISPLAY statement transfers data from the program to the CRT screen. Refer to Chapter 7, "Procedure Division in the Nucleus" for another format of the DISPLAY statement.

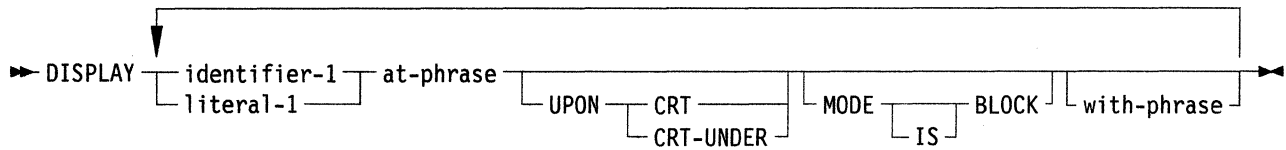
General Format

The following figures shows the format of the DISPLAY statement:

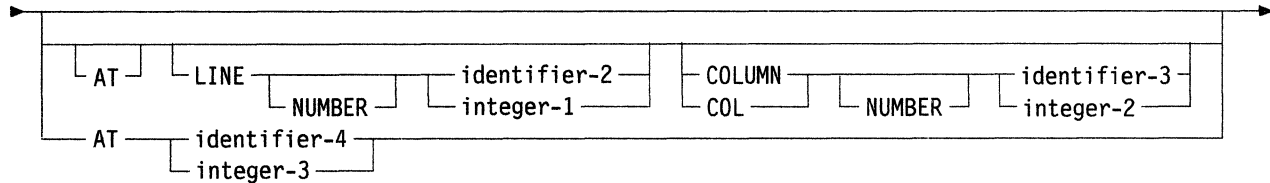
Format 1



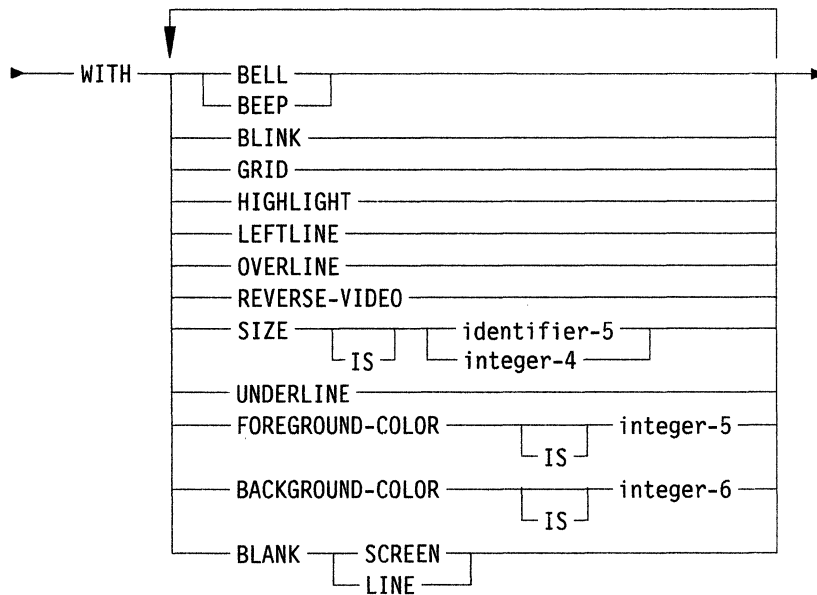
Format 2



where at-phrase is:



where with-phrase is:



Syntax Rules

The following syntax rules apply to the DISPLAY statement:

Both Formats

1. The LINE and COLUMN phrases can appear in any order.
2. integer-3 must be 4 or 6 digits long.

Format 1

3. identifier-3 must be 4 or 6 digits long.

Format 2

4. identifier-4 must be 4 or 6 digits long.
5. A DISPLAY statement with an operand that is an identifier or a literal is treated as Format 2 if it has:
 - a. An AT phrase
 - b. An UPON phrase with the CRT or CRT-UNDER option
 - c. A WITH phrase
 - d. Or a MODE IS BLOCK phrase.
 - e. No UPON phrase but the CONSOLE IS CRT clause is specified in the SPECIAL-NAMES paragraph.

If it has the UPON phrase with the CONSOLE option, or if it has no UPON phrase and the CONSOLE IS CRT phrase is not specified in the SPECIAL-NAMES paragraph, it is treated as the standard ANSI DISPLAY statement. Refer to "DISPLAY Statement" on page 7-32.

6. The phrases following the identifier can be in any order.
7. If identifier-1 or literal-1 is not specified, then the MODE IS BLOCK and WITH phrases are not allowed.
8. The attributes GRID, LEFTLINE, and OVERLINE are provided for compatibility. They are accepted syntactically but have no effect at run time.
9. The attribute BLINK is accepted syntactically but will have effect only on a terminal that supports blinking.
10. When the CRT-UNDER option is specified, the output of that DISPLAY statement will be underlined.

General Rules

The following general rules apply to the DISPLAY statement:

Both Formats

1. The AT phrase gives the absolute address on the screen where the DISPLAY operation is to start.
2. If integer-3 is 4 digits long, the first two digits specify the line, and the second two specify the column. If integer-3 is 6 digits long, the first three digits specify the line, while the second three specify the column.
3. Certain combinations of line and column numbers have special meanings, as follows:
 - a. Until the column comes within range, out-of-range column values are reduced by the line length and the line value is incremented.
 - b. Out-of-range line values cause the screen to scroll up one line. The effect is the same as if the line number of the bottom line had been specified.
 - c. If the line and column numbers given are both zero, the DISPLAY starts at the position following the position where the preceding Format 1 or Format 2 DISPLAY operation finished. Column 1 of each line is considered to follow the last column of the previous line.
 - d. If the line number is zero but a nonzero column number is specified, the DISPLAY starts at the specified column on the line following the line where the preceding Format 1 or 2 DISPLAY operation finished.
 - e. If the column number is zero but a nonzero line number is specified, the DISPLAY starts on the specified line at the column following the column where the preceding Format 1 or 2 DISPLAY operation finished.

Format 1

4. If identifier-3 is 4 digits long, the first two digits specify the line, and the second two specify the column. If identifier-3 is 6 digits long, the first three digits specify the line, and the second three specify the column.
5. This format of the DISPLAY statement displays screen items, which are defined within the screen section of the program, and allows full access to the enhanced screen-handling facilities.

Format 2

6. If identifier-4 is 4 digits long, the first two digits specify the line, the second two the column. If identifier-4 is 6 digits long, the first three digits specify the line, the second three the column.
7. Part of this statement can be repeated to allow the display of several data items. If the first identifier has no AT phrase, it begins at line 1, column 1. Each subsequent data item begins at the current cursor position at left after the previous data item.
8. If identifier-1 is a group item and there is no MODE IS BLOCK phrase, then those elementary subordinate items which have names other than FILLER are displayed. They are displayed simultaneously on the screen in the order their descriptions appear in the DATA DIVISION, and separated by the lengths of the FILLER items in the group. For this purpose, the first position on a line is regarded as immediately following the last position on the previous line.
9. The MODE IS BLOCK phrase indicates that the identifier is to be treated as an elementary item. Thus, even if it is a group item it is displayed as one item.
10. If no identifier or literal is present, the DISPLAY operation moves the cursor without actually displaying any data.
11. If the identifier is one of the following figurative constants it has a special effect as follows:
 - a. SPACE clears from the specified cursor position to the end of the screen
 - b. LOW-VALUE moves the cursor to the specified position
 - c. ALL X"01" clears from the specified cursor position to the end of the line
 - d. ALL X"02" clears the whole screen
 - e. ALL X"07" sounds the bellIf the identifier is a figurative constant that is not listed above and the SIZE option is not specified, one occurrence of its value is displayed.
12. If the SIZE option is specified for a figurative constant that has no special effect, then the figurative constant is displayed as many times as necessary to reach the length specified in the size option. However, if the display wraps around to a new line, it starts again at the beginning of the figurative constant.
13. If both the BLANK SCREEN and FOREGROUND-COLOR options are specified, the designated color becomes the default foreground color.
14. If both the BLANK SCREEN and BACKGROUND-COLOR options are specified, the designated color becomes the default background color.

Screen-Handling Sample Program

The following example shows a complete screen-handling program:

```
IDENTIFICATION DIVISION
PROGRAM-ID SCDEM01.

* THIS PROGRAM DEMONSTRATES SCREEN-HANDLING. IT
* SHOWS, FOR DEMONSTRATION PURPOSES ONLY, HOW
* PART OF A PROGRAM FOR HANDLING A BANK ACCOUNT
* MIGHT BE WRITTEN: IT ACCEPTS A PASSWORD, AND
* THEN REPEATEDLY ACCEPTS DETAILS OF CHECKS.
* NEITHER THE PASSWORD NOR THE CHECK DETAILS ARE
* ACTUALLY PROCESSED: ONLY THE SCREEN-HANDLING
* CODE IS GIVEN. FOR BOTH, THE PROGRAM USES
* FORMAT 1 OF THE ACCEPT AND DISPLAY VERBS, WHICH
* REFERENCES SCREEN ITEMS. THE MESSAGE ASKING THE
* OPERATOR WHETHER TO CONTINUE DEMONSTRATES
* FORMAT 2, WHICH REFERENCES DATA ITEMS.

ENVIRONMENT DIVISION.
SPECIAL-NAMES.
    CURSOR IS CURSOR-POSITION

DATA DIVISION.
WORKING-STORAGE SECTION.

01  CURSOR-POSITION.
    02  CURSOR-LINE    PIC 99.
    02  CURSOR-COL    PIC 99.

01  DATE-COLS.
    02  DAY-COL       PIC 99 VALUE 56.
    02  MONTH-COL    PIC 99 VALUE 59.
    02  YEAR-COL     PIC 99 VALUE 62.

*
* AREAS TO CONTAIN DATA FROM FORMS ON THE SCREEN.

01  P-PASSWORD      PIC X(20) VALUE SPACES.

01  S-CHECK.
    02  S-TO         PIC X(15) OCCURS 4.
    02  S-FOR        PIC X(10).
    02  S-REFNO      PIC X(10).
    02  S-AMOUNT     PIC S9(6)V99.
    02  S-DATE.
        03  S-DAY     PIC 99.
        03  S-MONTH   PIC 99.
        03  S-YEAR    PIC 9999.
```

*
* AREA FOR FORMAT 2 ACCEPT/DISPLAY USED FOR
* "CONTINUE" MESSAGE

01 GO-ON.

02 GO-MESSAGE PIC X(55) VALUE
"DO YOU WANT TO ENTER ANOTHER? (Y FOR YES,
N TO FINISH):".

02 FILLER PIC X.

01 GO-ON-RED.

02 FILLER PIC X(55).

02 GO- ON-REPLY PIC X.

SCREEN SECTION.

01 PASSWORD-FORM.

02 BLANK SCREEN.

02 VALUE "PLEASE ENTER PASSWORD <".

02 PIC X(20) TO P-PASSWORD, REQUIRED, SECURE.

02 VALUE ">".

01 CHECK-FORM.

03 BLANK SCREEN.

03 LINE 2, COLUMN 35, VALUE "CHECK".

03 LINE 4.

03 OCCURS 4.

04 PIC X FROM "<", LINE, COLUMN 10.

04 PIC X (15) USING S-TO.

04 PIC X FROM ">".

03 VALUE "RECIPIENT NAME AND ADDRESS",
LINE + 2, COLUMN 6.

03 VALUE "<", LINE 14, COLUMN 10.

03 PIC ZZZ, ZZ9.99 USING S-AMOUNT,
REQUIRED.

03 VALUE ">".

03 VALUE "AMOUNT", LINE 15, COLUMN - 9.

03 VALUE "<", LINE 5, COLUMN 40.

03 PIC X(10) USING S-FOR.

03 VALUE ">".

03 VALUE " FOR", LINE 6, COLUMN - 9.

03 VALUE "<", LINE 5, COLUMN 60.

03 PIC X(10) USING S-REFNO, FULL.

03 VALUE ">".

03 VALUE "REFERENCE", LINE 6, COLUMN - 9.

03 VALUE "DATE:", LINE 14, COLUMN 46.
03 VALUE "<", COLUMN DAY-COL.
03 PIC ZZ USING S-DAY, PROMPT IS "D",
AUTO, REQUIRED.
03 VALUE "/", COLUMN MONTH-COL.
03 PIC ZZ USING S-MONTH, PROMPT IS
"M", AUTO, REQUIRED.
03 VALUE "/", COLUMN YEAR-COL.
03 PIC ZZZZ USING S-YEAR, PROMPT IS "Y",
FULL, REQUIRED.

03 VALUE "> ".

01 ERROR MESSAGES, BLINK.
02 CLEAR-MSG LINE 24, BLANK LINE.
02 MSG1 LINE 24, VALUE "INVALID DAY".
02 MSG2 LINE 24, VALUE "INVALID MONTH".
02 MSG3 LINE 24, VALUE "INVALID YEAR".

PROCEDURE DIVISION.

START-UP.

DISPLAY PASSWORD-FORM.

ACCEPT PASSWORD-FORM.

* CODE TO CHECK THE PASSWORD WOULD BE WRITTEN HERE.

INITIALIZE-CHECK.

MOVE SPACES TO CURSOR-POSITION.

DISPLAY CLEAR-MSG.

MOVE SPACES TO S-CHECK.

MOVE ZERO TO S-AMOUNT S-DAY S-MONTH S-YEAR.

DISPLAY CHECK-FORM.

ACCEPT-CHECK.

ACCEPT CHECK-FORM.

```
VALIDATE-DATE.  
  MOVE 14 TO CURSOR-LINE.  
  IF S-DAY > 31  
    DISPLAY MSG1  
    ADD DAY-COL 1 GIVING CURSOR-COL  
    GO TO ACCEPT-CHECK  
  ELSE IF S-MONTH > 12  
    DISPLAY MSG2  
    ADD MONTH-COL 1 GIVING CURSOR-COL  
    GO TO ACCEPT-CHECK  
  ELSE IF S-YEAR < 1900 OR > 2100  
    DISPLAY MSG3  
    ADD YEAR-COL 1 GIVING CURSOR-COL  
    GO TO ACCEPT-CHECK  
  ELSE  
    DISPLAY CLEAR-MSG.
```

```
* CODE TO PROCESS THE CHECK, PERHAPS WITH FURTHER  
* VALIDATION OF THE DATE, WOULD BE WRITTEN HERE.
```

```
NEXT-CHECK.  
  MOVE SPACE TO GO-ON-REPLY.  
  DISPLAY GO-ON AT 2401.  
  ACCEPT GO-ON-RED AT 2401.  
  IF GO-ON-REPLY = "y" OR "Y" GO TO INITIALIZE-  
  CHECK.  
  IF GO-ON-REPLY = "n" OR "N" STOP RUN.
```

```
GO TO NEXT CHECK.
```

Appendix A. Ryan-McFarland Syntax Supplement

Introduction	A-3
Reserved Words	A-3
Identification Division - The PROGRAM-ID Paragraph	A-3
Environment Division	A-3
ASSIGN Clause	A-3
ORGANIZATION Clause	A-5
Data Division	A-6
VALUE OF LABEL Clause	A-6
Default Sign Representation	A-6
USAGE Clause	A-7
Procedure Division	A-8
Literals as CALL Parameters	A-8
EXIT PROGRAM Statement	A-8
Bound Checking	A-8
Size Allocation for Index Data Items	A-8
STOP RUN Statement	A-8
Nonstandard Operations on Alphanumeric Data Items	A-9
PERFORM Statement	A-9
Procedure Names	A-9
REWRITE on Line-Sequential Files	A-9
OPEN and CLOSE on SEQUENTIAL Files	A-9
ACCEPT Statement	A-10
DISPLAY Statement	A-11
File I-O Status Codes	A-11

Introduction

This appendix lists **syntax** which is accepted by your COBOL system for compatibility with revision 2.0 of Ryan-McFarland (RM) COBOL. Many features found in RM/COBOL exist already in AIX VS COBOL. These are documented in the main body of this manual. Some of the compatibility syntax listed in this appendix is identical to AIX VS COBOL syntax, but behaves differently if you set the RM system directive when you submit your source code to your COBOL system. See the *User's Guide* for details of the RM directive.

Reserved Words

The following non-standard reserved words are given as reserved words when the RM directive is given:

BEEP	BINARY
BLINK	COMP-1
COMP-3	COMP-6
COMPUTATIONAL-1	COMPUTATIONAL-3
COMPUTATIONAL-6	CONVERT
ECHO	EOL
EOS	ERASE
HIGH	POS
PROMPT	PRINT
REVERSE	TAB
UNLOCK	UPDATE

Identification Division - The PROGRAM-ID Paragraph

The feature of RM/COBOL that allows a data-name to be the same as the PROGRAM-ID is not supported in the AIX VS COBOL language.

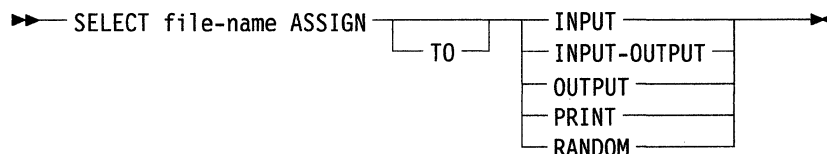
Environment Division

The following information describes the Ryan-McFarland compatibility syntax of the Environment Division of an AIX VS COBOL program.

ASSIGN Clause

Format

The following format of the ASSIGN clause is supported:



General Rule

The words INPUT, OUTPUT, INPUT-OUTPUT, PRINT, and RANDOM are treated as documentary.

ORGANIZATION Clause

Format

The following format of the ORGANIZATION clause is supported:

► ORGANIZATION IS BINARY SEQUENTIAL ◄

General Rules

The following rules apply to the ORGANIZATION clause:

1. This clause is treated as equivalent to ORGANIZATION IS SEQUENTIAL.

However, if you specify the ANSI parameter with the RM system directive when you submit your source code to the AIX VS COBOL system, this clause is treated as equivalent to ORGANIZATION IS LINE-SEQUENTIAL.

2. If you do not specify an ORGANIZATION clause for a file, then ORGANIZATION IS LINE-SEQUENTIAL is assumed, providing you set the RM system directive when you submit your source code to the AIX VS COBOL system.

However, if you specify the ANSI parameter with the RM system directive when you submit your source code to the AIX VS COBOL system, then ORGANIZATION IS SEQUENTIAL is assumed.

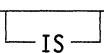
Data Division

The following information describes the Ryan-McFarland syntax for items found in the Data Division of AIX VS COBOL programs:

VALUE OF LABEL Clause

Format

The following clause is supported in an FD entry:

▶— VALUE OF LABEL —  — literal —▶

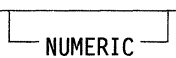
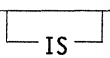
This clause is treated as documentary.

Length of Nonnumeric Literals

In the Data Division, nonnumeric literals up to 2,047 characters long are allowed, if the RM system directive is set.

Default Sign Representation

The following figure shows the format for the default sign representation:

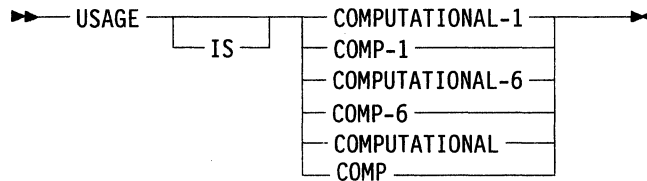
▶—  — SIGN —  — TRAILING SEPARATE —▶

This syntax is assumed in the Special-Names paragraph if the RM system directive is set. If you specify RM with the ANSI parameter, this phrase is not assumed.

USAGE Clause

Format

The following format of the USAGE clause is supported:



Syntax Rules

The following syntax rules apply to the USAGE clause:

1. COMP and COMPUTATIONAL are synonymous.
2. COMP-1 and COMPUTATIONAL-1 are synonymous.
3. COMP-6 and COMPUTATIONAL-6 are synonymous.

General Rules

The following general rules apply to the USAGE clause:

1. The COBOL system allocates a 2 byte signed binary data item capable of holding values in the range $-32k$ to $+32k$, for each data item declared as USAGE COMP-1 in the source program, regardless of its picture string. Each USAGE COMP-1 data item is treated as though it had a standard AIX VS COBOL picture string of S9(4) COMP.
2. COMP-6 data items are treated as the standard AIX VS COBOL COMP format. If, as a result of this, less data space is allocated to each item than would be under the RM/COBOL system, the space is padded with null bytes (X'00').
3. USAGE IS COMPUTATIONAL means the same as USAGE IS DISPLAY.

Procedure Division

The following information describes the Ryan-McFarland compatibility syntax for items in the Procedure Division.

Literals as CALL Parameters

Literals can be used as parameters to CALL statements.

EXIT PROGRAM Statement

The EXIT PROGRAM statement closes all files that the exited subprogram has opened if RM system directive is set.

However, if you specify the ANSI parameter with the RM directive, the EXIT PROGRAM statement does not close these files.

Bound Checking

Table subscripts are not bound checked.

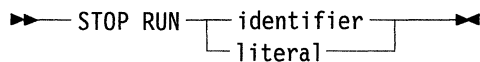
Size Allocation for Index Data Items

Two bytes are allocated to index data items instead of the normal four when RM system directive is set.

STOP RUN Statement

Format

The following format of the STOP RUN statement is supported:



Syntax Rules

The following rules apply to the STOP RUN statement:

1. *identifier* must be a numeric integer data item.
2. *literal* must be a numeric integer.

General Rule

The value held in the numeric integer data item or the value of the literal is placed in a special register RETURN-CODE.

Nonstandard Operations on Alphanumeric Data Items

Alphanumeric values can be stored in numeric-data items, but will be right justified (as is normal with a numeric receiving item) and space-filled on the left. For example, if "AB" is moved to an item described as PIC 9(5), the value stored will be " AB".

PERFORM Statement

The AIX VS COBOL system normally uses a stack for handling PERFORM statements, while RM COBOL associates a return address with a specific procedure name. As a result, with RM COBOL all end points of PERFORM statements are active until they are used, whereas under AIX VS COBOL only the end point of the innermost current PERFORM statement is active at any one time.

However, if the PERFORM(RM) directive is set, AIX VS COBOL will handle PERFORM statements in the way RM COBOL does. Refer to the *User's Guide*.

Procedure Names

Programs can contain procedure names that are the same as data-names.

REWRITE on Line-Sequential Files

The REWRITE statement can be used on sequential files, providing the new record is the same length as the original.

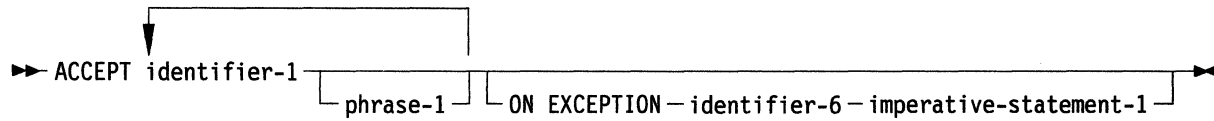
OPEN and CLOSE on SEQUENTIAL Files

The RM/COBOL language permits OPEN and CLOSE statements to be specified with NO REWIND on a sequential file. In the AIX VS COBOL language, such statements are treated as documentary only.

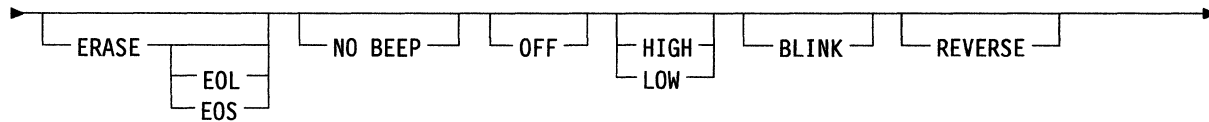
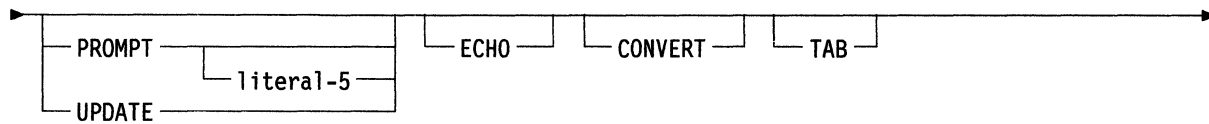
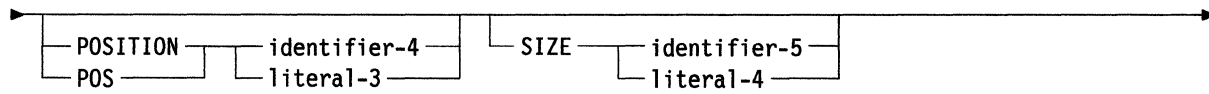
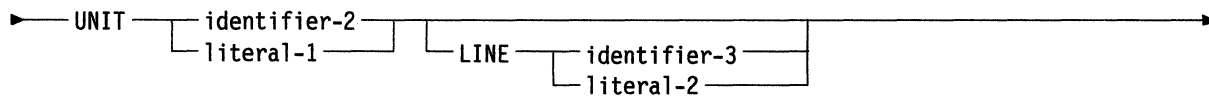
ACCEPT Statement

Format

The following format for the ACCEPT statement is supported:



where phrase-1 is:



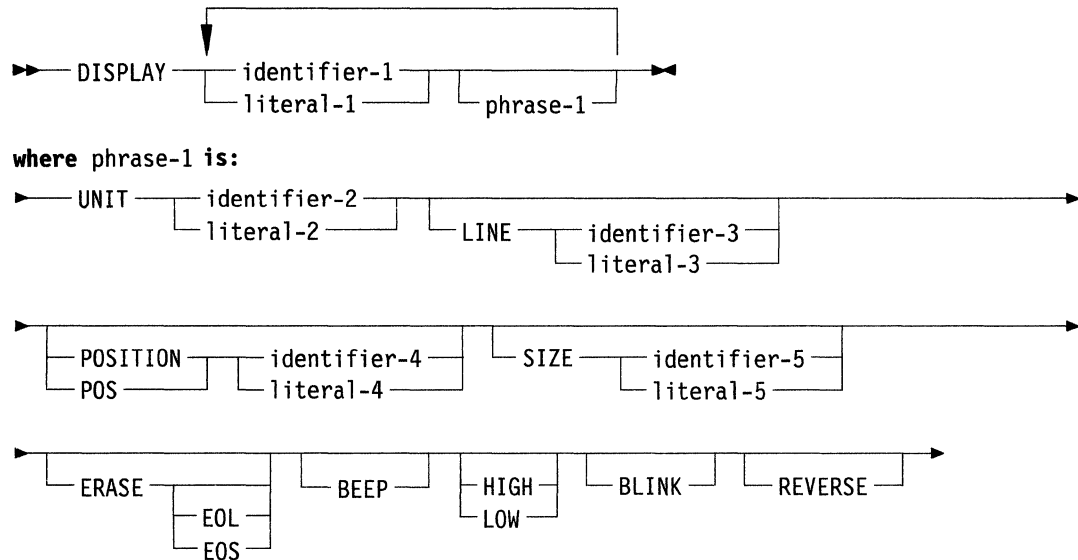
General Rule

This format is treated as equivalent to Format 2 of the ACCEPT statement in the Screen-Handling module of AIX VS COBOL. However, multiple operands are allowed.

DISPLAY Statement

Format

The following format for the DISPLAY statement is supported:



General Rule

This format is treated as equivalent to Format 2 of the DISPLAY statement in the Screen-Handling module of AIX VS COBOL.

File I-O Status Codes

AIX VS COBOL I-O status codes are mapped onto their RM equivalents as shown in Table A-1.

AIX VS COBOL I-O Codes		RM/COBOL I-O Codes		Meaning
Key 1	Key 2	Key 1	Key 2	
9	2	9	1	File not open when access attempted
9	4	9	4	Illegal file name
9	5	9	5	Illegal device specification
9	6	9	0	Attempt to write to a file open input
9	8	9	0	Attempt to read from a file open output
9	12	9	2	Attempt to open a file already open
9	13	9	4	File not found

AIX VS COBOL I-O Codes		RM/COBOL I-O Codes		Meaning
Key 1	Key 2	Key 1	Key 2	
9	17	9	7	Record error : probably zero length
9	18	9	0	Attempt to read part of record
9	19	9	0	Rewrite error : open mode or access mode wrong
9	30	9	4	File system is read only
9	31	9	4	Not owner of this file
9	35	9	4	Incorrect access permission
9	37	9	4	File access denied
9	38	9	5	Disk not compatible
9	39	9	5	File not compatible
9	41	9	8	Corrupt index file
9	43	9	8	File information missing for indexed file
9	47	9	8	Indexed structure overflow
9	65	9	3	File locked
9	66	2	2	Attempt to add duplicate record key to indexed file
9	67	9	1	Indexed file not open
9	68	9	9	Record locked
9	69	9	8	Illegal argument to ISAM module
9	71	9	8	Bad indexed file format
9	72	9	8	End of indexed file
9	73	9	8	No record found in indexed file
9	74	9	8	No current record in indexed file
9	75	9	4	Name of indexed file too long
9	78	9	8	Illegal key description in indexed file
9	81	2	2	Key already exists in indexed file
9	100	9	0	Invalid file operation
9	101	9	0	Illegal operation on indexed file
9	104	9	4	Null file name used in a file operation.
9	138	9	3	File closed with lock -- cannot open
9	139	9	0	Record length or key inconsistent
9	141	9	2	File already open -- cannot open
9	142	9	1	File not open -- cannot close
9	143	9	0	Rewrite/delete not after successful read

Table A-1 (Page 3 of 3). Mapping of File I-O Status Codes				
AIX VS COBOL I-O Codes		RM/COBOL I-O Codes		Meaning
Key 1	Key 2	Key 1	Key 2	
9	146	9	6	No current record (sequential read)
9	147	9	0	Wrong open/access mode (read/start)
9	148	9	0	Wrong open/access mode (write)
9	149	9	0	Wrong open/access mode (rewrite/delete)
9	151	9	0	Random read on sequential file
9	152	9	0	Rewrite on file not open I-O
9	158	9	0	Rewrite on line-sequential file
9	182	9	0	Console in/out open in wrong mode
9	183	9	4	Attempt to open line-sequential file for I-O
9	188	9	4	File name too large
9	194	3	4	File size too large
9	195	9	0	Delete/rewrite not preceded by a read
9	196	9	8	Record number is too large in a relative file
9	210	9	3	File is closed with lock
9	any other	3	0	

Locked Records

The following happens if a record is found to be locked:

1. If the file has an applicable declaratives section and it has a file status value associated with it:
 - Processing is transferred to the declaratives section
 - The record area contains the correct record for the read
 - The I-O status code indicates the "record locked" condition.
2. If the file has no applicable declaratives section or file status associated with it, the locked record is continually retried until its lock is released.

Appendix B. Data General Syntax Supplement

Introduction	B-3
Long User-Defined Names	B-3
Environment Division	B-4
Switch Names	B-4
File Name On Disk	B-4
DATA SIZE Clause	B-4
INDEX SIZE Clause	B-4
Duplicate Alternate Keys	B-5
Alternate Keys	B-5
I-O Control Entry	B-5
Data Division	B-6
VALUE Clause	B-6
SCREEN SECTION	B-6
Procedure Division	B-7
ACCEPT Statement	B-7
CALL Statement	B-8
COPY INDEXED Statement	B-8
DISPLAY Statement	B-8
File Sharing Syntax	B-8
OPEN Statement	B-8
READ Statement	B-8

Introduction

This appendix lists syntax that is accepted by your COBOL system purely for compatibility with revision 1.3 of Data General Interactive COBOL. Many features found in Data General Interactive COBOL already exist in AIX VS COBOL. These features are documented in the main body of this manual. Some of the compatibility syntax listed in this appendix is identical to AIX VS COBOL syntax, but behaves differently if you set the DG system directive when you submit your source code to your COBOL system. See the *User's Guide* for details of the DG directive.

Long User-Defined Names

Data-names and procedure names longer than 30 characters are allowed. However, only the first 30 characters are significant. The cross referencing listing and ANIMATOR functions only handle the first 30.

Environment Division

This section describes the Data General compatibility syntax used for items in the Environment Division.

Switch Names

In addition to the switches 0 to 8 supported by IBM AIX VS COBOL, another form of switch name can be used. It is an uppercase letter in the range A to Z.

These letters are mapped to switches 0 to 25. At run time, you should use the appropriate digit on your command line and not the letter itself.

For example, to turn on switch J, you would enter:

```
+9 filename
```

after the run command. You cannot, however, specify a run-time switch in a CALL statement, as in:

```
CALL "PROG.INT/A"
```

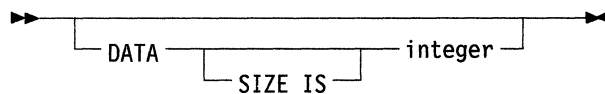
because this is not supported.

File Name On Disk

When you assign a file to disk, the file on disk is named in uppercase letters, regardless of whether the name you specify is in uppercase or lowercase letters.

DATA SIZE Clause

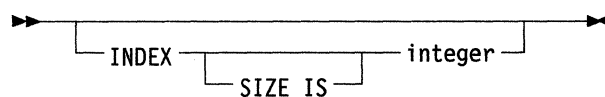
In a SELECT statement, you can use:



It is treated as documentary.

INDEX SIZE Clause

In a SELECT statement for a relative or indexed file, you can use:



It is treated as documentary.

Duplicate Alternate Keys

You can have duplicate alternate keys in an indexed file regardless of whether or not you have specified the `DUPLICATES` phrase.

Alternate Keys

Alternate keys can occupy the same area as primary keys.

I-O Control Entry

The `SAME AREA` phrase is treated as equivalent to the `SAME RECORD AREA` phrase.

Data Division

This section describes the Data General syntax for items in the Data Division.

VALUE Clause

A numeric literal can be used in a VALUE clause in the data description of a nonnumeric data item.

SCREEN SECTION

The HIGHLIGHT option when used with TO or USING items highlights all the unprotected areas of the display screen.

Procedure Division

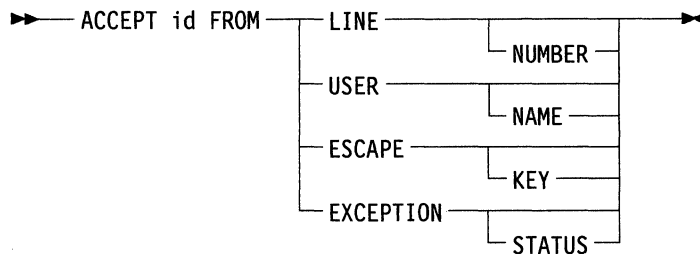
This section describes the Data General compatibility syntax for items in the Procedure Division.

ACCEPT Statement

This section describes the compatibility format and general rules for the ACCEPT statement.

Format

Format 2 of the ACCEPT statement in the Nucleus is enhanced with additional options:



General Rules

The following general rules apply to the ACCEPT statement:

1. The value returned from the ACCEPT FROM LINE NUMBER phrase is always numeric.

For users operating under the AIX operating system, the value returned by the ACCEPT FROM LINE NUMBER clause denotes the device number of the terminal attached to standard input. The device number is of the form *major device number, minor device number*, such as 1, 3, and this is returned as a decimal number. For example, a device number of 1, 3 would be returned as 259 (1 times 256, plus 3).

2. The FROM USER NAME option returns a user id number on AIX systems.
3. The FROM EXCEPTION STATUS option contains a three character code that identifies the type of exception condition that has occurred during the execution of a CALL or CALL PROGRAM statement. Its PICTURE is 9(3).

If the FROM EXCEPTION STATUS is to be examined it should be done immediately after the CALL or CALL PROGRAM, before execution of any other statements. File input-output operations will alter the exception status value making the value undefined. The CALL ... ON OVERFLOW or CALL ... ON EXCEPTION will cause the value of the EXCEPTION STATUS to be undefined.

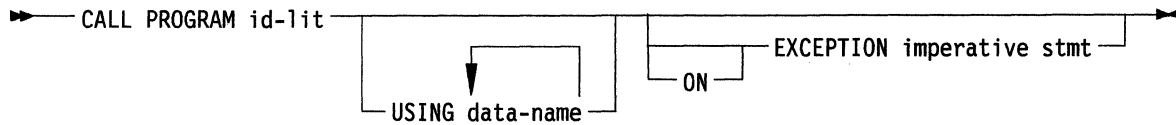
4. ESCAPE KEY contains the two-digit code generated by a termination key.

CALL Statement

This section describes the compatibility format and general rule for the CALL statement.

Format

The following example shows the format for the CALL statement:



General Rule

This format is treated as equivalent to a CHAIN statement.

COPY INDEXED Statement

The COPY statement can be followed by the word INDEXED. The word INDEXED is treated as documentary.

DISPLAY Statement

The DISPLAY statement with the WITH NO ADVANCING option is changed to be as in DG COBOL.

File Sharing Syntax

The default lock mode for both Indexed Sequential Access Method (ISAM) and relative files is MANUAL WITH LOCK ON MULTIPLE RECORDS.

OPEN Statement

The OPEN verb can be followed by the keyword EXCLUSIVE. It is treated as equivalent to the WITH LOCK phrase.

READ Statement

The file name and the noise word RECORD can be followed by the keyword LOCK. It causes the READ statement to acquire a lock on the record read. This does not apply to line sequential files.

Appendix C. Microsoft Syntax Supplement

Introduction	C-3
Compatibility with Microsoft COBOL	C-3
Dialect Controlling Directives	C-3
Summary of Syntactic Differences	C-3
Special Registers LIN and COL	C-3
Environment Division	C-4
Data Division	C-4
Procedure Division	C-5
Problem Determination	C-8
General	C-9
Environment Division	C-9
Data Division	C-10
Procedure Division	C-10
File Input and Output	C-11
Screen-Handling	C-12
Extension Subroutines	C-13
Documentation	C-13

Introduction

This appendix lists syntax that is accepted by your COBOL system purely for compatibility with Microsoft COBOL. Many features found in Microsoft COBOL already exist in AIX VS COBOL. These features are documented in the main body of this manual. Some of the compatibility syntax listed in this appendix is identical to AIX VS COBOL syntax, but behaves differently if you set the IBM-MS system directive when you submit your source code to your COBOL system. See the *User's Guide* for details of the IBM-MS directive.

Compatibility with Microsoft COBOL

IBM AIX VS COBOL is fully compatible with Microsoft COBOL V1.0, and the corresponding IBM PC COBOL V1.0. Some minor differences between the product and Microsoft COBOL V2.2 are described below.

Dialect Controlling Directives

The compiler has a default setting of NOIBM-MS, which switches off all specific Microsoft V2.2 syntax and semantics. To get maximum compatibility with the Microsoft V2.2 compiler use the following directives:

MS(2)	OEXT(COB)	NOOSVS	MF	NORW
NOANS85	NOBOUND	OLDINDEX	NOTRUNC	AUTOLOCK
NOOPTIONAL-FILE		PERFORM-TYPE (RM)	SIGN (EBCDIC)	

For details of the effects of these, see the *User's Guide*.

Summary of Syntactic Differences

This section indicates the areas where the syntax and semantics may differ between Microsoft V2.2 and AIX VS COBOL.

Special Registers LIN and COL

LIN and COL are intended to be used for specifying the line number and column number, respectively, of the current cursor position on the screen. Together they form part of the position-spec phrase which is supported in the ACCEPT, DISPLAY and EXHIBIT statements documented later in this appendix.

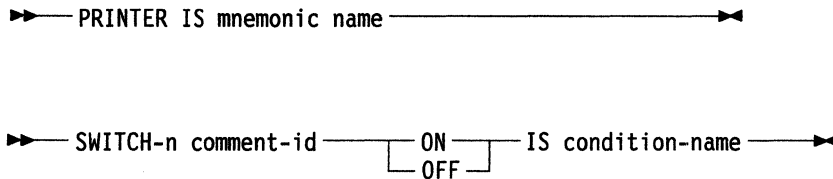
The format of the LIN and COL is PIC S9(4) COMP. They are used like ordinary data items, except that they must not be declared; they are automatically declared by the system. The programmer should set them to the desired values before using them in the position-spec phrase.

Note that COL, the abbreviation for COLUMN, is not available as a Micro Focus reserved word if the IBM-MS directive is set.

Environment Division

The Special-Names Paragraph

The following two clauses are supported:



Where: n is an integer in the range 0 to 8. The default setting is OFF.

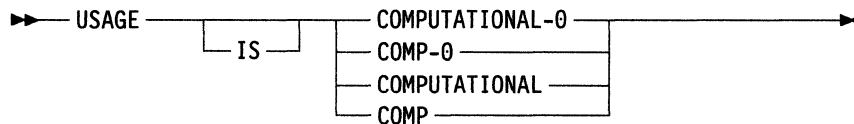
The *User's Guide* contains full details on how you can set a run-time switch on.

Data Division

The USAGE Clause

Format

The following format of the `USAGE` clause is supported:



Syntax Rules

1. `COMP` and `COMPUTATIONAL` are synonymous.
2. `COMP-0` and `COMPUTATIONAL-0` are synonymous.

General Rules

1. `COMPUTATIONAL` data items are treated as external decimal items.
2. `USAGE IS COMPUTATIONAL` means the same as `USAGE IS DISPLAY`.
3. `USAGE IS COMP-0` is allowed only with numeric items. If an item is described with a `PICTURE` no longer than `S9(5)`, the effect is a data item whose description is `PIC S9(4) USAGE COMP`; that is, a 2-byte signed binary item. Otherwise, the effect is a data item with the `PICTURE` clause actually specified and with `USAGE DISPLAY`.
4. You must use the `NOTRUNC` option to cause correct behavior for `COMP-0` items.

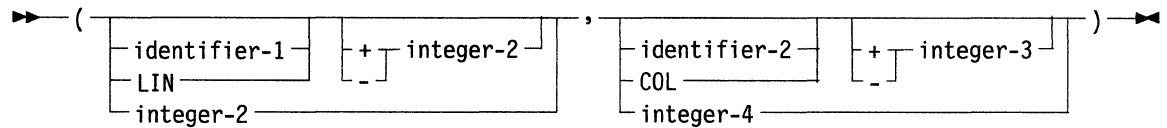
Procedure Division

The Position-Spec Phrase

Function

The position-spec phrase specifies the screen position in the ACCEPT, DISPLAY and EXHIBIT statements described in this appendix.

Format



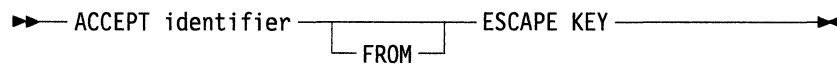
Syntax Rule

The comma shown in the above format is mandatory.

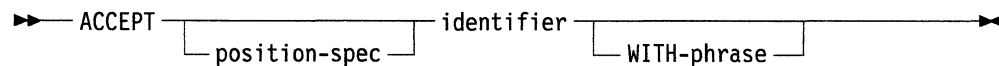
The ACCEPT statement

The following two formats of the ACCEPT statement are supported:

Format 1



Format 2



General Rules

Format 2

1. The ACCEPT operation treats a group item as an elementary item: it accepts the item itself and not its subordinate elementary items.
2. If the screen position at which the first operand is to appear is not specified, the default is the current cursor position.
3. Rules 1 and 2 above also apply to Format 2 of the ACCEPT statement as described in "ACCEPT Statement" on page 18-55.

4. The WITH-phrase options and their synonyms are:

AUTO/AUTO-SKIP
BACKGROUND-COLOR
BELL/BEEP
BLINK
REVERSE-VIDEO
RIGHT-JUSTIFY
SECURE/NO-ECHO
SIZE
SPACE-FILL
TRAILING-SIGN
UNDERLINE
UPDATE
ZERO-FILL

These options also apply to Format 2 of the ACCEPT statement as documented in "ACCEPT Statement" on page 18-55.

The options listed below differ from standard AIX VS COBOL, as follows:

- a. **UPDATE**—If the update option is not specified, a data item for data entry is displayed initially as spaces. (If the IBM-MS directive is not set, the display of initial data is a configuration option.)
If the UPDATE option is specified, initial data of the data item for data entry is displayed.
- b. **PROMPT**—This option need not be specified to display prompt characters.
- c. **LENGTH-CHECK**—If this option is specified, it causes an implicit **EMPTY-CHECK** option, so that the operator must enter something.
- d. **FOREGROUND-COLOR** and **BACKGROUND COLOR**—Integer-1 in these clauses specifies the foreground color and background color of the screen item respectively, and can be defined by a value from 0 to 15, as follows:

0	black	8	grey
1	blue	9	light blue
2	green	10	light green
3	cyan	11	light cyan
4	red	12	light red
5	magenta	13	light magenta
6	brown	14	yellow
7	white	15	high-intensity white

On a color screen, if the foreground color option specifies an integer whose value is from 8 to 15, this is equivalent to specifying an integer from 0 to 7 and specifying the **HIGHLIGHT** option. On a monochrome screen, this is equivalent to simply specifying the **HIGHLIGHT** option.

On a color screen, if the background color option specifies an integer whose value is from 8 to 15, this is equivalent to specifying an integer from 0 to 7 and specifying the **BLINK** clause. On a monochrome screen, this is equivalent to simply specifying the **BLINK** option.

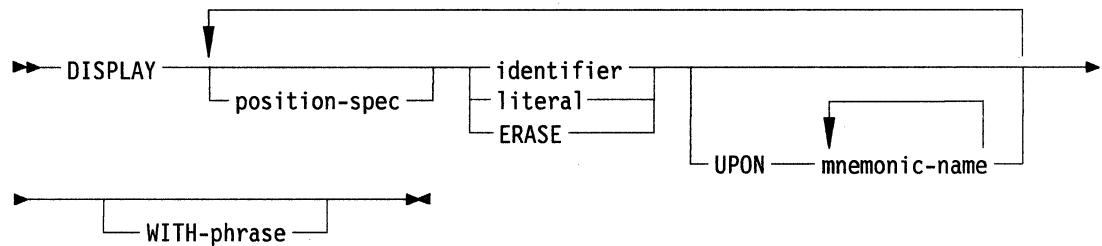
If both the foreground color and the background color are specified with integers greater than 7, the foreground color will have the HIGHLIGHT effect. The background will not have the BLINK effect.

- e. The attribute BLINK is accepted syntactically. However, it will only have effect at run time if the display being used supports blinking.

The DISPLAY Statement

Format

The following format of the DISPLAY statement is supported:



General Rules

1. The DISPLAY operation treats a group item as an elementary item: it displays the item itself and not its subordinate elementary items.
2. If the screen position at which the first operand is to appear is not specified, the default is the current cursor position.
3. Rules 1 and 2 above also apply to Format 2 of the DISPLAY statement as described in "DISPLAY Statement" on page 18-61.
4. If ERASE is specified, the screen is cleared from the current cursor position onwards.

The following two rules are additional rules which apply to Format 2 of the DISPLAY statement as documented in "DISPLAY Statement" on page 18-61.

5. The WITH phrase options and their synonyms are:

BACKGROUND-COLOR
BELL/BEEP
BLANK
BLINK
FOREGROUND-COLOR
HIGHLIGHT
REVERSE-VIDEO
SIZE
UNDERLINE

-
6. FOREGROUND-COLOR and BACKGROUND-COLOR—Integer-1 in these clauses specifies the foreground color and background color of the screen item respectively, and can be defined by a value from 0 to 15, as follows:

0	black	8	grey
1	blue	9	light blue
2	green	10	light green
3	cyan	11	light cyan
4	red	12	light red
5	magenta	13	light magenta
6	brown	14	yellow
7	white	15	high-intensity white

On a color screen, if the foreground color option specifies an integer whose value is from 8 to 15, this is equivalent to specifying an integer from 0 to 7 and specifying the HIGHLIGHT option. On a monochrome screen, this is equivalent to simply specifying the HIGHLIGHT option.

On a color screen, if the background color option specifies an integer whose value is from 8 to 15, this is equivalent to specifying an integer from 0 to 7 and specifying the BLINK clause. On a monochrome screen, this is equivalent to simply specifying the BLINK option.

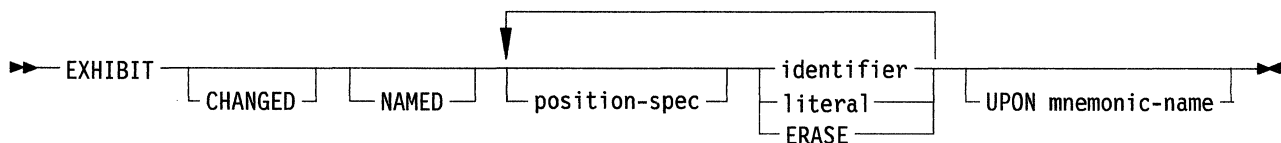
If both the foreground color and the background color are specified with integers greater than 7, the foreground color will have the HIGHLIGHT effect. The background will not have the BLINK effect.

7. BLINK is always accepted syntactically, but will have effect only on a terminal that supports blinking.

The EXHIBIT Statement

Format

The following format of the EXHIBIT statement is supported:



General Rule

ERASE clears the screen from the current cursor position onward.

Problem Determination

The following sections indicate some problems you may encounter, and some suggested resolutions. There are some minor cases when the current compiler will produce errors when encountering missing periods or spelling errors (for example, SOURCE COMPUTER in place of SOURCE-COMPUTER will cause an error message to be issued.)

General

Problem

Source code appears to extend beyond column 72 and an error is issued by the compiler. This will occur if the standard TAB settings on the Microsoft V2.2 compiler have been changed (either by patching the run time or by using the /O directive).

Solution

Remove all tab settings from the source files.

Difference

Error messages and numbers returned by the compiler and the run time are different. This should present no problems, but it is something you should be aware of.

Environment Division

Problem

EJECT IS ... (in SPECIAL-NAMES) is rejected by the compiler.

Solution

Replace the word EJECT by the word FORMFEED.

Problem

The compiler gives error on SPECIAL-NAMES paragraph header.

Solution

This occurs if the SPECIAL-NAMES paragraph is not inside the ENVIRONMENT DIVISION.

Problem

SEGMENT-LIMIT IS ... This clause is now treated as documentary.

Solution

This should cause no problems, but it is worth noting that segmentation is an outdated feature of COBOL (the function it performs is really an operating system function) and has been marked by ANSI as an obsolete feature.

Data Division

Problem

Items with PIC clauses like `-.ZZ` fail compilation. These were allowed by the Microsoft V2.2 compiler but are not valid ANSI COBOL syntax.

Solution

Change the PIC field description.

Problem

REDEFINES will not compile if it refers to a data item that does not immediately precede the redefining item of the same level. For example.

```
03 A.  
...  
03 B.  
...  
03 C REDEFINES A.
```

This is illegal because C is not the next item of the same level after A.

Solution

Move the redefinition so it is the next item of the same level.

Procedure Division

Problem

The CHAIN statement is not fully supported, and use of this syntax should be avoided.

Solution

Restructure the application to use the CALL statement.

Problem

It is not possible to sound the BELL by displaying an item that contains hexadecimal 7.

Solution

Use the RTE subprogram call X"E5" in place of the display.

Problem

The screen is not cleared automatically at the start of a run.

Solution

Use the RTE subprogram call X"E4" to clear the screen.

Problem

READY TRACE and RESET TRACE statements do not have any effect.

Solution

Use the compiler directive TRACE to enable processing of these statements at run time. The default is that they are treated as documentary by the compiler, to avoid the generation of the extra code needed for their support.

Problem

A DECLARATIVES statement is rejected by the compiler if it occurs after the paragraph header.

Solution

The DECLARATIVES statement must immediately follow the SECTION header, rather than the paragraph header.

File Input and Output

Problem

OPEN EXTEND on a non-existent file will not create it.

Solution

Add the word OPTIONAL after the word SELECT for this file.

Problem

The WAIT clause on a READ behaves differently. This is ignored by the current compiler; instead, the file status is updated to indicate that a record is locked.

Solution

Check for the Record locked status and loop back to re-read the record.

Problem

Locking syntax is ignored in the START statement.

Solution

Use the appropriate locking semantics in the OPEN or READ statements and the SELECT statement for the file in question.

Problem

Relative records are not deleted if they are rewritten with the first byte containing LOW-VALUES.

Solution

Using this feature of the Microsoft V2.2 system was discouraged in the manual. The program logic should be replaced with a DELETE statement.

Problem

Incorrect filenames are being used by programs.

Solution

All filenames must be terminated by a space, where the Microsoft V2.2 compiler allowed termination with a null byte.

Other Differences

The structure of Microsoft COBOL files is different from the Micro Focus file format used in the IBM AIX VS COBOL system.

Screen-Handling

Problem

COLUMN PLUS 1 does not give the correct intervening space under some circumstances. For instance, the lines

```
02 LINE 1 COLUMN 1...
02      COLUMN PLUS 1...
```

will not put a space in column 2, while

```
02 LINE 1 COLUMN 1...
02 LINE 1 COLUMN PLUS 1...
```

will work correctly.

Solution

Add an explicit LINE clause.

Problem

Alphabetic fields do not allow non-alphabetic characters to be accepted into them.

Solution

Change the PIC A(n) of the field to PIC X(n).

Other Differences

- Integer NUMERIC field are ACCEPTed differently; digits are inserted from left to right. Only NUMERIC-EDITED fields insert digits from right to left.
- DISPLAY ... ERASE does not clear the screen attributes, which it does in Microsoft COBOL.
- Cursor left/right will move between fields in an ACCEPT. In Microsoft COBOL these keys move only within a field.
- Cursor up/down behaves differently between fields.
- Ctrl-B does not move to a previous field as it does in Microsoft COBOL.

Extension Subroutines

The following Microsoft V2.2 Extension subroutines do not exist in AIX VS COBOL:

COMMAND	CURPOS	EXCODE
EXIST	KBDVAIL	LOCASE
REMOVE	RENAME	UPCASE

Documentation

When this manual refers to COMP fields, these references are intended to be to BINARY fields rather than the Microsoft V2.2 USAGE IS COMP, since the latter are treated in the same way as USAGE DISPLAY items.



Appendix D. Reserved Word List

Introduction

Table D-1 on page D-4 lists the reserved words in AIX VS COBOL.

An x in a column indicates that the word is reserved in that dialect or module. A blank indicates that the word is not reserved there. A * in a column indicates that the word is reserved. However, these starred items are not supported in this implementation.

Not all reserved words from *imported* syntaxes (for example, OSVS and VSC2) are supported in AIX VS COBOL, but they are still treated as reserved words to prevent them from being used as user-defined words. This prevents them from causing problems if compiled with a mainframe compiler that supports those dialects.

The syntax for the optional ANSI module Communications is accepted by AIX VS COBOL. However, Communications is not supported at run time.

Key:

AN85	= ANSI85 required modules
85RW	= ANSI85 Report Writer Module
85CM	= ANSI85 Communications Module
85DB	= ANSI85 Debug Module
85SG	= ANSI85 Segmentation Module
SAA	= SAA COBOL CPI
OSVS	= OSVS dialect of AIX VS COBOL
VSC2	= VSC2 dialect of AIX VS COBOL
MF	= MF extensions in AIX VS COBOL

Table D-1 (Page 1 of 9). Reserved Words									
Reserved Word	AN85	85RW	85CM	85DB	85SG	SAA	OSVS	VSC2	MF
(x	x	x	x	x	x	x	x	x
)	x	x	x	x	x	x	x	x	x
*	x	x	x	x	x	x	x	x	x
**	x	x	x	x	x	x	x	x	x
+	x	x	x	x	x	x	x	x	x
-	x	x	x	x	x	x	x	x	x
. (period)	x	x	x	x	x	x	x	x	x
/	x	x	x	x	x	x	x	x	x
;	x	x	x	x	x	x	x	x	x
<	x	x	x	x	x	x	x	x	x
<=	x	x	x	x	x	x	x	x	x
=	x	x	x	x	x	x	x	x	x
>	x	x	x	x	x	x	x	x	x
>=	x	x	x	x	x	x	x	x	x
:	x							x	
'	x	x	x	x	x	x	x	x	x
"	x	x	x	x	x	x	x	x	x
\$	x	x	x	x	x	x	x	x	x
,	x	x	x	x	x	x	x	x	x
ACCEPT	x					x			
ACCESS	x					x			
ACTUAL							*		
ADD	x					x			
ADDRESS								x	
ADVANCING	x					x			x
AFTER	x					x			
ALL	x					x			
ALPHABET	x					x			
ALPHABETIC	x					x			
ALPHABETIC-LOWER	x					x		x	
ALPHABETIC-UPPER	x					x		x	
ALPHANUMERIC	x					x		x	
ALPHANUMERIC-EDITED	x					x		x	
ALSO	x					x			
ALTER	x					x			
ALTERNATE	x					x			
AND	x					x			
ANY	x					x		x	
APPLY							x		
ARE	x					x			
AREA	x					x			
AREA-VALUE (1)									x
AREAS	x					x			
ASCENDING	x					x			
ASSIGN	x					x			
AT	x					x			
AUTHOR	x					x			
AUTO									x
AUTO-SKIP									x
AUTOMATIC									x
BACKGROUND-COLOR									x
BACKGROUND-COLOUR									x
BACKWARD									x
BEEP									x
BEFORE	x					x			
BEGINNING							x	x	
BELL									x
BINARY	x					x		x	
BLANK	x					x			
BLINK									x
BLOCK	x					x			
BOTTOM	x					x			
BY	x					x			
C01							x		
C02							x		
C03							x		
C04							x		

Table D-1 (Page 2 of 9). Reserved Words									
Reserved Word	AN85	85RW	85CM	85DB	85SG	SAA	OSVS	VSC2	MF
C05							x		
C06							x		
C07							x		
C08							x		
C09							x		
C10							x		
C11							x		
C12							x		
CALL	x					x			
CANCEL	x					x		*	
CBL									
CD			x						
CF		x							
CH		x							
CHAIN									x
CHAINING									x
CHANGED							x		x
CHARACTER	x					x			
CHARACTERS	x					x			
CLASS	x					x		x	
CLOCK-UNITS	x					x			
CLOSE	x					x			
COBOL	x					x			
CODE		x							
CODE-SET	x	x				x			
COL (2)									x
COLLATING	x					x			
COLUMN		x							x
COM-REG								*	
COMMA	x					x			
COMMAND-LINE							x		x
COMMIT									x
COMMON	x					x		x	
COMMUNICATION	x					x			
COMP	x					x			
COMP-0									x
COMP-3							x	x	x
COMP-4							x	x	
COMP-5									x
COMP-X									x
COMPUTATIONAL	x					x			
COMPUTATIONAL-0									x
COMPUTATIONAL-3							x	x	x
COMPUTATIONAL-4							x	x	
COMPUTATIONAL-5									x
COMPUTATIONAL-X									x
COMPUTE	x					x			
CONFIGURATION	x					x			
CONSOLE							x		x
CONTAINED								x	
CONTAINS	x					x			
CONTENT	x							x	
CONTINUE	x					x		x	
CONTROL		x							
CONTROLS		x							
CONVERTING	x					x		x	
COPY	x					x			
CORE-INDEX							x		
CORR	x					x			
CORRESPONDING	x					x			
COUNT	x		x			x			
CRT									x
CRT-UNDER									x
CSP							x		
CURRENCY	x						x		
CURRENT-DATE							x		

Table D-1 (Page 3 of 9). Reserved Words

Reserved Word	AN85	85RW	85CM	85DB	85SG	SAA	OSVS	VSC2	MF
CURSOR									x
DATA	x					x			
DATE	x		x			x			
DATE-COMPILED	x					x			
DATE-WRITTEN	x					x			
DAY	x					x			
DAY-OF-WEEK	x					x		x	
DBCS						x		x	
DE		x							
DEBUG-CONTENTS				x					
DEBUG-ITEM				x					
DEBUG-LINE				x					
DEBUG-NAME				x					
DEBUG-SUB-1				x					
DEBUG-SUB-2				x					
DEBUG-SUB3				x					
DEBUGGING	x					x			
DECIMAL-POINT	x					x			
DECLARATIVES	x					x			
DELETE	x					x			
DELIMITED	x					x			
DELIMITER	x					x			
DEPENDING	x					x			
DESCENDING	x					x			
DESTINATION			x						
DETAIL		x							
DISABLE			x						
DISK									x
DISP							x		
DISPLAY	x					x			
DISPLAY-1						x		x	x
DISPLAY-ST							*		
DIVIDE	x					x			
DIVISION	x					x			
DOWN	x					x			
DUPLICATES	x					x			
DYNAMIC	x					x			
EGCS								x	
EGI			x						
EJECT						x	x	x	
ELSE	x					x			
EMI			x						
EMPTY-CHECK									x
ENABLE			x						
END	x					x			
END-ACCEPT									x
END-ADD	x					x			
END-CALL	x					x			
END-COMPUTE	x					x			
END-DELETE	x					x			
END-DIVIDE	x					x			
END-EVALUATE	x					x			
END-IF	x					x			
END-MULTIPLY	x					x			
END-OF-PAGE	x					x			
END-PERFORM	x					x			
END-READ	x					x			
END-RECEIVE			x						
END-RETURN	x					x			
END-REWRITE	x					x			
END-SEARCH	x					x			
END-START	x					x			
END-STRING	x					x			
END-SUBTRACT	x					x			
END-UNSTRING	x					x			
END-WRITE	x					x			

Table D-1 (Page 4 of 9). Reserved Words									
Reserved Word	AN85	85RW	85CM	85DB	85SG	SAA	OSVS	VSC2	MF
ENDING ENTER ENTRY	x					x	x x	x x	
ENVIRONMENT EOP EQUAL	x x x					x x x			
ERASE ERROR ESCAPE	x					x			x x
ESI EVALUATE EVERY	x x		x			x x		x	
EXAMINE EXCEPTION EXCESS-3	x					x	x	x	*
EXCLUSIVE EXEC EXECUTE									x x x
EXHIBIT EXIT EXTEND	x x					x x	x		x
EXTERNAL FALSE FD	x x x					x x x		x x	x
FILE FILE-CONTROL FILE-ID	x x					x x			x
FILE-LIMIT FILE-LIMITS FILLER	x					x	x x		
FINAL FIRST FIXED	x	x x				x			x
FOOTING FOR FOREGROUND-COLOR	x x					x x			x
FOREGROUND-COLOUR FROM FULL	x					x			x x
GENERATE GIVING GLOBAL	x x	x				x x		x	
GO GOBACK GREATER	x x					x x	x	x	
GRID GROUP HEADING		x x							x
HIGH-VALUE HIGH-VALUES HIGHLIGHT	x x					x x			x
ID IDENTIFICATION IF	x x					x x	x	x	
IN INDEX INDEXED	x x x					x x x			
INDICATE INITIAL INITIALIZE	x x	x	x			x		x	
INITIATE INPUT INPUT-OUTPUT	x x	x				x x			
INSPECT INTO INVALID	x x x					x x x			

Table D-1 (Page 5 of 9). Reserved Words

Reserved Word	AN85	85RW	85CM	85DB	85SG	SAA	OSVS	VSC2	MF
I-O	x					x			
I-O-CONTROL	x					x			
IS	x					x			
INSTALLATION	x					x			
JUST	x					x			
JUSTIFIED	x					x			
KANJI								x	
KEPT	x								x
KEY									x
KEYBOARD									x
LABEL	x					x			
LAST		x							
LEADING	x					x			
LEAVE							*		
LEFT	x					x			
LEFT-JUSTIFY									x
LEFTLINE									x
LENGTH			x						x
LENGTH-CHECK									x
LESS	x					x			
LIMIT		x							
LIMITS		x							
LINAGE	x					x			
LINAGE-COUNTER (3)	x					x			
LINE	x					x			
LINE-COUNTER		x							
LINES	x					x			
LINKAGE	x					x			
LOCK	x					x			x
LOW-VALUE	x					x			
LOW-VALUES	x					x			
MANUAL									x
MEMORY	x					x			
MERGE	x					x			
MESSAGE			x						
MODE	x					x			
MODULES	x					x			
MORE-LABELS							*	*	
MOVE	x					x			
MULTIPLE	x					x			
MULTIPLY	x					x			
NAME									x
NAMED							x		x
NATIVE	x					x			
NEGATIVE	x					x			
NEXT	x					x			
NO	x					x			
NO-ECHO									x
NOMINAL							*		
NOT	x					x	*		
NOTE									x
NULL								x	
NULLS								x	
NUMBER		x							x
NUMERIC	x					x			
NUMERIC-EDITED	x					x		x	
OBJECT-COMPUTER	x					x			
OCCURS	x					x			
OF	x					x			
OFF	x					x			
OMITTED	x					x			
ON	x					x	x		
OPEN	x					x			
OPTIONAL	x					x			
OR	x					x			
ORDER	x					x		x	
ORGANIZATION	x					x			

Table D-1 (Page 6 of 9). Reserved Words									
Reserved Word	AN85	85RW	85CM	85DB	85SG	SAA	OSVS	VSC2	MF
OTHER	x					x		x	
OTHERWISE							x		
OUTPUT	x					x			
OVERFLOW	x					x			
OVERLINE									x
PACKED-DECIMAL	x					x			
PADDING	x					x		x	
PAGE	x					x			
PAGE-COUNTER		x							
PASSWORD							x	x	
PERFORM	x					x			
PF		x							
PH		x							
PIC	x					x			
PICTURE	x					x			
PLUS		x							x
POINTER	x					x		x	
POSITION	x					x			
POSITIONING							x		
POSITIVE	x					x			x
PREVIOUS									x
PRINT-SWITCH							x		x
PRINTER									x
PRINTER-1									x
PRINTING		x							
PROCEDURE	x					x			
PROCEDURES									
PROCEED	x			x		x			
PROCESSING								x	
PROGRAM	x					x			
PROGRAM-ID	x					x			
PROMPT									x
PROTECTED									x
PURGE			x					x	
QUEUE			x						
QUOTE	x					x			
QUOTES	x					x			
RANDOM	x					x			
RANGE									x
RD		x							
READ	x					x			
READY							x		x
RECEIVE			x						
RECORD	x					x			
RECORD-OVERFLOW							x		
RECORDING							x		
RECORDS	x					x		x	x
REDEFINES	x					x			
REEL	x					x			
REFERENCE	x							x	
REFERENCES									
RELATIVE	x			x		x			
RELEASE	x					x			
RELOAD							*	*	
REMAINDER	x					x			
REMARKS							x		
REMOVAL	x					x			
RENAMES	x					x			
REORG-CRITERIA							x		
REPLACE	x					x		x	
REPLACING	x					x			
REPORT		x							
REPORTING	x	x							
REPORTS		x							
REQUIRED									x

Table D-1 (Page 7 of 9). Reserved Words

Reserved Word	AN85	85RW	85CM	85DB	85SG	SAA	OSVS	VSC2	MF
REREAD							*		
RERUN	x					x			
RESERVE	x					x			
RESET		x					x	x	x
RETURN	x					x			
RETURN-CODE							x	x	
REVERSE-VIDEO									x
REVERSED	x					x			
REWIND	x					x			
REWRITE	x					x			
RF		x							
RH		x							
RIGHT	x					x			
RIGHT-JUSTIFY									x
ROLLBACK									x
ROUNDED	x					x			
RUN	x					x			
S01							x		
S02							x		
SAME	x					x			
SCREEN									x
SD	x					x			
SEARCH	x					x			
SECTION	x					x			
SECURE									x
SECURITY	x					x			
SEGMENT			x						
SEGMENT-LIMIT					x				
SEEK							*		
SELECT	x					x			
SELECTIVE							*		
SEND			x						
SENTENCE	x					x			
SEPARATE	x					x			
SEQUENCE	x					x			
SEQUENTIAL	x					x			
SERVICE							*	*	
SET	x					x		x	
SHIFT-IN								*	
SHIFT-OUT								*	
SIGN	x					x			
SIZE	x					x			
SKIP1							x	x	
SKIP2							x	x	
SKIP3							x	x	
SORT	x					x			
SORT-CONTROL								x	
SORT-CORE-SIZE							x	x	
SORT-FILE-SIZE							x	x	
SORT-MERGE	x					x			
SORT-MESSAGE							x	x	
SORT-MODE-SIZE							x	x	
SORT-RETURN							x	x	
SOURCE		x	x						
SOURCE-COMPUTER	x					x			
SPACE	x					x			
SPACE-FILL									x
SPACES	x					x			
SPECIAL-NAMES	x					x			
STANDARD	x					x			
STANDARD-1	x					x			
STANDARD-2	x					x			
START	x					x			
STATUS	x		x			x			
STOP	x					x			

Table D-1 (Page 8 of 9). Reserved Words									
Reserved Word	AN85	85RW	85CM	85DB	85SG	SAA	OSVS	VSC2	MF
STORE							*		
STRING	x					x			
SUB-QUEUE-1			x						
SUB-QUEUE-2			x						
SUB-QUEUE-3			x						
SUBTRACT	x					x			
SUM		x							
SUPPRESS		x					x	x	
SYMBOLIC	x		x						
SYNC	x					x			
SYNCHRONIZED	x					x			
SYSIN							x		
SYSIPT							x		
SYSLIST							x		
SYSLST							x		
SYSOUT							x		
SYSPNCH							x		
SYSPUNCH							x		
TABLE			x						
TALLY							x	x	
TALLYING	x					x			
TAPE	x					x			
TERMINAL			x						
TERMINATE		x							
TEST	x					x		x	
TEXT			x						
THAN	x					x			
THEN	x					x	x	x	x
THROUGH	x					x			
THRU	x					x			
TIME	x		x			x			
TIME-OF-DAY							x		
TIMES	x					x			
TITLE						x		x	
TO	x					x			
TOP									
TOTALED							*		
TOTALING							*		
TRACE							x	x	x
TRACK-AREA							*		
TRACK-LIMIT							*		
TRACKS							*		
TRAILING	x					x			
TRAILING-SIGN									x
TRANSFORM							x		
TRUE	x					x		x	
TYPE		x							
UNDERLINE									x
UNIT	x					x			
UNLOCK									x
UNSTRING	x					x			
UNTIL	x					x			
UP	x					x			
UPDATE									x
UPON	x					x			
USAGE	x					x			
USE	x					x			
USER									x
USING	x					x			
VALUE	x					x			
VALUES	x					x			
VARIABLE									x
VARYING	x					x			
WHEN	x					x			
WHEN-COMPILED							x	x	

Table D-1 (Page 9 of 9). Reserved Words									
Reserved Word	AN85	85RW	85CM	85DB	85SG	SAA	OSVS	VSC2	MF
WITH WORDS WORKING-STORAGE	x x x					x x x			
WRITE WRITE-ONLY ZERO	x x					x x	x		
ZERO-FILL ZEROES ZEROS	x x					x x			x
Notes:									
(1) AREA-VALUE is a reserved word only if the RESERVE...AREA clause is used.									
(2) COL is not a reserved word if the IBM-MS dialect is used.									
(3) LINAGE-COUNTER is a reserved word only if the LINAGE clause is used.									

Appendix E. Obsolete Language Elements

Introduction	E-3
List of Obsolete Language Elements	E-3

Introduction

This appendix lists those language elements that are marked as obsolete in the ANSI X3.23-1985 COBOL specification. The language elements listed here:

- Are to be deleted at the next revision of the standard COBOL specification
- Will not be enhanced, modified, or maintained.

List of Obsolete Language Elements

- Associating the figurative constant *ALL literal* where *literal* is longer than one character with a data item that is numeric or numeric edited.
- The paragraphs AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, and SECURITY within the IDENTIFICATION DIVISION.
- The MEMORY SIZE clause of the OBJECT-COMPUTER paragraph.
- The RERUN clause of the I-O-CONTROL paragraph.
- The MULTIPLE FILE TAPE clause of the I-O-CONTROL paragraph.
- The LABEL RECORDS clause in the file description entry.
- The VALUE OF clause in the file description entry.
- The DATA RECORDS clause in the file description entry.
- The ALTER statement.
- The KEY phrase within the DISABLE statement.
- The KEY phrase within the ENABLE statement.
- The ENTER statement.
- The option to include *procedure-name* in the GO TO statement.
- The REVERSED phrase within the OPEN statement.
- The STOP *literal* statement.
- The segmentation module.
- The debug module.

Glossary

abbreviated combined relation condition. The combined condition that results from the explicit omission of a common subject or a common subject and common relation operator in a consecutive sequence of relation conditions.

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 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).

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 established the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

arithmetic expression. An identifier or 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 operator. A single character, or a fixed two character combination that belongs to the following set:

Character	Meaning
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

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, which allows tape file processing in accordance with the following standards:

- American National Standard Code for Information Interchange, X3.4-1968
- American National Standard Magnetic Tape Labels for Information Interchange, X3.27-1969
- American National Standard Recorded Magnetic Tape for Information Interchange (800 CPI NRZI), X3.22-1967

ASCII control characters. Characters listed in the ASCII code table.

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:

- 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.
- During the execution or a RETURN statement, when no next logical record exists for the associated sort or merge file.
- During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

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 operation 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 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 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.

buffer. A portion of storage used to hold input or output data temporarily.

byte. A string consisting of a certain number of bits, usually eight, treated as a unit, and representing a character.

CD-name. A user-defined word that names an MCS interface area described in a communication description entry within the COMMUNICATION SECTION of the Data Division.

called program. A program that is the object of a CALL statement combined at object time with the calling program to produce a run-unit. The term is synonymous with subprogram.

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.

chained program. A program that is the object of a CHAIN statement.

chaining program. A program that executes a CHAIN to another program.

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 comment entry.

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 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.

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 SAA COBOL character set consists of the 77 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 (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

COBOL system directing statement. A statement, beginning with a directing verb, that causes your COBOL system to take a specific action during creation of the intermediate code.

COBOL word. See *word*.

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 that, despite being directly contained within another program, may be called from any program directly or indirectly contained in that other program.

communication description entry. An entry in the COMMUNICATION SECTION of the Data Division that is composed of the level indicator CD, followed by a cd-name, and then followed by a set of clauses as required. The entry describes the interface between the Message Control System (MCS) and the COBOL program.

Communication Device. A mechanism (hardware or hardware/software) capable of sending data to a queue or receiving data from a queue or both. This mechanism may be a computer or a peripheral device. One or more programs containing communication description entries and residing within the same computer define one or more of these mechanisms.

COMMUNICATION SECTION. The section of the Data Division that describes the interface areas between the MCS and the program, composed of one or more CD description entries. See *message control system*.

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. The SAA compiler directing statements are COPY, EJECT, SKIP 1/2/3, TITLE, and USE.

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. 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.

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 implementer defined switch or device. When condition name is used in 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.

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 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.

CONFIGURATION SECTION. A section of the Environment Division that describes overall specifications of source and object programs.

connective. A reserved word that is used to do the following:

- Associate a data-name, paragraph-name, condition-name, or text-name with the reserved word's qualifier.
- Link two or more operands written in a series.
- Form conditions (logical connectives). See *logical operator*.

CONSOLE. A COBOL environment name associated with the operator console.

constant-name. A user-defined word assigned as the name of a fixed value.

contiguous items. Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

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.

CRT. An output device by which an operator can receive visual data.

currency sign. The character \$ of the COBOL character set.

currency symbol. The character defined by the CURRENCY SIGN clause in the SPECIAL NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign.

current record. In file processing the record that is available in the record area associated with a file.

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 dictionary. A table, built by your COBOL system and held in memory, that contains information on each user-defined name.

Data Division. One of the four main components of a COBOL program. The Data Division describes the data to be processed by the object program: files to be used and the records con-

tained within them; internal Working Storage records that will be needed; data to be made available in more than one program in the run-unit.

data item. A unit of data (excluding literals) defined by the COBOL program.

data-name. A user defined word that names a data item described in a data item described in a data description entry. When used in the general formats, data-name represents a word that must not be subscripted or qualified unless specifically permitted by the rules of the format.

DBCS (Double Byte Character Set). See *Double Byte Character Set (DBCS)*.

debugging line. A debugging line is any line with a D in the indicator area of the line.

DEBUGGING SECTION. A section that contains a USE FOR DEBUGGING statement.

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.

declarative sentence. A compiler directing sentence consisting of a single USE statement terminated by the separator period.

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.

destination. The symbolic identification of the receiver of a transmission from a queue.

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 in a COBOL program are:

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

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. Since each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software which 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.

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.

editing character. A single character or fixed two character combination belonging to the following set:

Character	Meaning
b	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
\$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	slant (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.

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. For example:

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. 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. Valid environment names for SAA COBOL are: SYSIN, SYSOUT, CONSOLE, C01, CSP, and UPSI-0 through UPSI-7. 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.

execution time. See *run time*.

execution time environment. See *run time environment*.

explicit scope terminator. A reserved word which 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.

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.

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 known as *zoned decimal item*.)

external file connector. A file connector that is accessible to one or more object programs in the run-unit.

figurative constant. A compiler generated value referenced through the use of certain reserved words.

file. A collection of logical records.

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

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 Control. The name of an Environment Division paragraph in which the data files for a given source program are declared.

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 is used in the selection of the next record.

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.

format. A specific arrangement of a set of data.

group item. A data item that is composed of subordinate data items.

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.

Identification Division. One of the four main component parts of a COBOL program. The Identification Division identifies the source program and the object program. The Identification Division may include the following documentation: author name, installation, or date.

identifier. A syntactically correct combination of a data-name, with its qualifiers and subscripts, as required for uniqueness of reference, that names a data item or the value of the referenced data item. The rules for identifier associated with the general formats may, however, specifically, prohibit qualification or subscripting.

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.

implementer-name. A system-name that refers to a particular feature available on the implementer's computing system.

implicit scope terminator. A separator period that terminates the scope of any preceding unterminated statement, or a phrase of a statement that, by occurring, indicates the end of the scope of any statement contained within the preceding phrase.

implicit segment. A segment created by your COBOL system to control the size of code segments.

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 implementer.

index name. A user defined word that names an index associated with a specific table.

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.

indicator area. The leftmost parameter position of a COBOL source record, that indicates the use of the record.

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 and that provides information required for transmission and handling of data during execution of the object program.

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.

integer. A numeric literal or a numeric data item that does not include any digit positions to the right of the assumed decimal point. When the term integer appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero unless explicitly allowed by the rules of that format.

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 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 0011 1111. (Also known as packed decimal.)

internal file connector. A file connector that is accessible only to one object program in a run-unit.

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 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.

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. 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 whose presence is required when the format in which the word appears is used in a source program.

language name. A system name that specifies a particular programming language.

level indicator. Two alphabetic characters that identify a specific type of file or a position in a hierarchy. The level indicators in the Data Division are: CD, FD, and SD.

level number. A user defined word, expressed as a two digit number, which indicates the hierarchical position of 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.

line-sequential file organization. A type of sequential file containing variable-length records in the format of text files produced by the host operating system.

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 implied by the ordered set of characters comprising the string.

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.

main program. In a hierarchy of programs and subroutines, the first program to receive control when the programs are run.

mass storage device. A device having a large storage capacity; for example, magnetic disk, magnetic drum.

MCS. See *Message Control System*.

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.

message. Data associated with an end-of-message indicator or an end-of-group indicator. (See *message indicators*.)

Message Control System (MCS). A communication control system that supports the processing of messages to and from terminal devices.

message count. The count of the number of complete messages that exist in the designated queue of messages.

message indicators. End-of-group indicator (EGI), end-of-message indicator (EMI), and end-of-segment indicator (ESI) are conceptual indications that serve to notify the MCS that a specific condition exists (end-of-group, end-of-message, end-of-segment).

Within the hierarchy of EGI, EMI, and ESI, an EGI is conceptually equivalent to an ESI, EMI, and EGI. An EMI is conceptually equivalent to an ESI and EMI. Thus, a segment may be terminated by an ESI, EMI, or EGI. A message may be terminated by an EMI or EGI.

message segment. Data that forms a logical subdivision of a message normally associated with an end-of-segment indicator. (See *message indicators*.)

mnemonic name. A user defined word that is associated in the Environment Division with a specified implementer name.

MVS/XA (Multiple Virtual Storage/Extended Architecture). An IBM operating system that manages multiple virtual address spaces in IBM processors operating in extended architecture mode. MVS/XA supports the 31 bit addressing mechanism of extended architecture mode and, thus, can manage an address space as large as 2 billion (that is, 2×10^9) bytes.

name. A word composed of not more than 30 characters that defines a COBOL operand.

native character set. The implementer defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

native collating sequence. The implementer 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.

next executable sentence. The next sentence to which control will be transferred after execution of the current statement is complete.

next executable statement. The next statement to which control will be transferred after execution of the current statement is complete.

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

noncontiguous items. Elementary data items, in the Working Storage and LINKAGE SECTIONS, that bear no hierarchic relationship to other data items.

nonnumeric item. A data item whose description permits its content to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

nonnumeric literal. A literal bounded by quotation marks. The string of characters may include any character in the computer's character set.

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 form 0 through 9, the decimal point, commas, the dollar sign, editing sign control symbols, plus other editing symbols.

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 also 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.

OBJECT-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the object program is executed, is described.

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.

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 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 multitasking operating system for the IBM Personal Computer family that allows you to run both DOS mode and OS/2 mode programs.

output field. A screen item whose description contains a FROM phrase.

output file. A file that is opened in either 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.

packed decimal item. See *Internal Decimal Item*.

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.
AUTHOR

INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.

The permissible paragraph headers in the Environment Division are:

SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
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.

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 item. An elementary data item to which a USAGE IS POINTER clause applies.

prime record key. A key whose contents uniquely identify a record within an indexed file.

procedure. A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure Division. One of the four main component parts of a COBOL program. The Procedure Division contains instructions for solving a problem. The Procedure Division may contain imperative statements, conditional statements, compiler directing statements, paragraphs, procedures, and sections.

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.

process. Any operation or combination of operations on data.

program name. In the Identification Division, a user defined word that identifies a COBOL library bounded by but not including, pseudo-text delimiters.

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 delimiters. Two contiguous equal sign (=) characters used to delimit pseudo-text.

punctuation character. A character that belongs to the following set:

Character	Meaning
,	comma
;	semicolon
.	period
"	quotation mark
(left parenthesis
)	right parenthesis
␣	space
=	equal sign

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.

queue. A logical collection (of messages, processes, print jobs) waiting to be transmitted or processed.

queue name. A symbolic name that tells the MCS the logical path by which a message or a portion of a completed message may be accessible in a queue.

Queued Sequential Access Method (QSAM). 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.

QSAM (Queued Sequential Access Method). See Queued Sequential Access Method.

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.

receiving item. A data item referred to in a TO or USING phrase in a PICTURE clause in the SCREEN SECTION.

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.

record key. A key whose contents identify a record within an indexed file. Within an indexed file in SAA COBOL, a record key is the prime record key.

record name. A user defined word that names a record described in a record description entry in the Data Division of a COBOL program.

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).

reel. A discrete portion of a storage medium, the dimensions of which are determined by each implementer, that contains part of a file, all of a file, or any number of files. The term is synonymous with unit and volume.

relation. See *relational operator*.

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*.)

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.

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.

routine. A set of statements in a program that causes the computer to perform an operation or series of related operations.

RTS. See *Run Time System*

run time. The time at which an object program is executed. The term is synonymous with object time.

Run Time Environment (RTE). A module that interprets intermediate code, provides various support services to native code and acts as an interface to the operating system.

Run Time System (RTS). See *Run Time Environment*.

run-unit. One or more object programs which interact with one another and which function, at object time, as an entity to provide problem solutions.

scope terminator. A COBOL reserved word that marks the end of certain Procedure Divi-

sion statements. It may be either explicit (END-ADD) or implicit (separator period).

screen description entry. An entry in the SCREEN SECTION of the Data Division that is composed of a level number, followed by an optional screen-name, and then by a set of screen clauses as required. This entry is very similar in structure to a data description entry, but while a data description entry declares areas in memory, a screen description entry declares areas on the screen.

screen item. A field on the screen to which the screen description entry assigns properties.

SCREEN SECTION. The last section within the Data Division, in which the layouts of the screen areas accessed in Format 2 ACCEPT and DISPLAY statements are defined.

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.
LINKAGE SECTION.
SCREEN SECTION.
REPORT SECTION.

In the Procedure Division, a section header is composed of 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.

segment-number. A user-defined word that classifies sections in the Procedure Division for segmentation. Segment-numbers may contain only characters from 0 through 9. A segment-number may be expressed either as a one- or a two-digit number.

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.

sending item. A data item referred to in a FROM or USING phrase in a PICTURE clause in the SCREEN SECTION.

sentence. A sequence of one or more statements, the last of which is terminated by a separator period.

separator. A punctuation character 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.

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
(simple condition)

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. The symbolic definition of the originator of a transmission to a queue.

SOURCE-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

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 absence of additional source program lines.

special character. A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign
*	asterisk
/	slant (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

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 registers. Certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of a specific COBOL feature.

split key. A concatenation of one or more data items within a record associated with that file-name. The split key can be referenced only in START and READ statements.

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.

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.

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*.

sub-queue. A logical hierarchical division of a queue.

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 -, which identifies a particular element in a table.

subscripted data-name. An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

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.

symbol function. The use of specified characters in the PICTURE clause to represent data types.

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.

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 name. A user defined word that identifies library text.

text word. A character or a sequence of contiguous characters following the indicator area

(column 7) in a COBOL library, source program, or in a pseudo-text which is:

- A separator, except for: space; a pseudo-text delimiter; and the opening and closing delimiters for nonnumeric 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 closing quotation mark that bound the literal.
- Any other sequence of contiguous COBOL characters except comment lines and the word COPY bounded by separators which is 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 the 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.

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.

UPSI switch. A program switch that performs the functions of a hardware switch. Eight switches 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.

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 occurrence data item. A table element that 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.

verb. A word that expresses an action to be taken by a COBOL compiler or object program.

Virtual Machine/System Product (VM/SP). 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.

Virtual Storage Access Method (VSAM). A high performance mass storage access method. Three types of data organization are available: entry sequenced data sets (ESDS), key sequenced data sets (KSDS), and relative record data sets (RRDS). Their COBOL equivalents are, respectively: sequential, indexed, and relative organizations.

VM/SP. See *Virtual Machine/System Product*.

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.

VSAM. See *Virtual Storage Access Method*

word. A character string of not more than 30 characters that forms a user defined word, a system name, or a reserved word.

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 both.

zoned decimal item. See *external decimal item*.

Index

A

abbreviated combined relation conditions 7-17
ACCEPT MESSAGE COUNT statement 15-20
ACCEPT statement 7-22, 18-55, A-10, B-7
ADD statement 7-24
ADDRESS Special Register 2-15, 11-17, 11-33
algebraic signs 2-19
alignment rules 2-19
alphabetic data rules 6-19
alphanumeric data rules 6-19
alphanumeric-edited data rules 6-19
ALTER statement 7-27, 16-7
alternate keys B-5
Apply clause
 CORE-INDEX 8-30
 RECORD-OVERFLOW 8-30
 REORG-CRITERIA 8-30
 WRITE-ONLY 8-30
arithmetic expressions 7-7
arithmetic operators 7-8
arithmetic statements 7-20
ASCII 5-8
ASSIGN clause 8-19, A-3
AT END condition 8-9, 8-11
AUTO clause 18-21

B

BACKGROUND-COLOR clause 18-22
BELL clause 18-24
BLANK clause 18-25
blank lines 2-53
BLANK WHEN ZERO clause 6-11, 18-26
blink 18-27, 18-58, 18-63
BLINK clause 18-27
BLOCK CONTAINS clause 8-39
body group presentation rules 14-38
bound checking A-8
byte order 2-27
byte storage mode 2-22

C

CALL statement 11-36, B-8
CANCEL statement 11-43
CHAIN statement 11-45
character set 2-4

character-strings 2-6
class condition 7-13
clause
 ASSIGN A-3
 AUTO 18-21
 BACKGROUND-COLOR 18-22
 BELL 18-24
 BLANK 18-25
 BLANK WHEN ZERO 6-11, 18-26
 BLINK 18-27
 BLOCK CONTAINS 8-39
 CODE 14-19
 CODE-SET 8-40
 COLUMN 18-28
 COLUMN NUMBER 14-45
 CONSOLE IS CRT 18-10
 CONTROL 14-20
 CRT STATUS 18-12
 CURSOR IS 18-11
 DATA RECORDS 8-42, 13-12
 DATA SIZE B-4
 Data-name 14-46
 Data-name or FILLER 6-12
 EXTERNAL 11-30
 FOREGROUND-COLOR 18-30
 FULL 18-32
 GLOBAL 11-31
 GRID 18-34
 GROUP INDICATE 14-47
 HIGHLIGHT 18-35
 INDEX SIZE B-4
 JUSTIFIED 6-13, 18-36
 LABEL RECORDS 8-43
 LEFTLINE 18-37
 LENGTH-CHECK 18-32
 LINAGE 8-44
 LINE 18-38
 LINE NUMBER 14-49
 NEXT GROUP 14-51
 OCCURS 12-5, 18-40
 ORGANIZATION A-5
 OVERLINE 18-42
 PAGE 14-22
 PICTURE 6-18, 18-43
 POINTER 6-40, 6-44
 PROMPT 18-45
 RECORD 8-47
 RECORD CONTAINS 13-13
 RECORDING MODE 8-50
 REDEFINES 6-29
 RENAMES 6-32
 REPORT 14-14

clause (*continued*)
 REQUIRED 18-46
 REVERSE-VIDEO 18-47
 SECURE 18-48
 SIGN 6-35, 14-53, 18-49
 SIZE 18-50
 SOURCE 14-55
 SUM 14-56
 SYNCHRONIZED 6-37
 TYPE 14-59
 UNDERLINE 18-51
 USAGE 6-39, 14-63, A-7
 USAGE IS INDEX 12-12
 VALUE 6-41, 14-64, 18-52, B-6
 VALUE OF 8-51
 WITH DEBUGGING MODE 17-5
 ZERO-FILL 18-53
 CLOSE statement
 DISP parameter 8-53
 in Report Writing facility 14-67
 COBOL
 See VS COBOL
 COBOL words 2-6
 CODE clause 14-19
 CODE-SET clause 8-40
 color 18-22, 18-30, 18-58, 18-62
 COLUMN clause 18-28
 COLUMN NUMBER clause 14-45
 combined and negated combined condition 7-16
 COMMAND-LINE 7-23, 7-33
 COMMAND-LINE function name 5-14
 comment-entries 2-15
 COMMIT statement 8-58
 common phrases 7-19
 common programs 11-7
 communication module
 Data Division
 description — complete entry
 skeleton 15-4
 introduction 15-4
 Procedure Division
 ACCEPT MESSAGE COUNT
 statement 15-20
 DISABLE statement 15-21
 ENABLE statement 15-23
 PURGE statement 15-25
 RECEIVE statement 15-26
 SEND statement 15-29
 Communications module 1-4, 15-3, D-3
 COMP-0 C-4
 comparison
 involving data items with USAGE
 POINTER 7-13
 involving index-names and/or index data
 items 7-12
 involving index-names and/or index data-
 items 12-13
 of nonnumeric operands 7-12
 comparison (*continued*)
 of numeric operands 7-12
 compile-time switch 17-4
 complex conditions 7-15
 COMPUTATIONAL-X or COMP-X format 2-28
 COMPUTATIONAL-0 C-4
 COMPUTATIONAL-1 or COMP-1 A-7
 COMPUTATIONAL-3, COMP-3, or
 PACKED-DECIMAL format 2-25
 COMPUTATIONAL-4 or COMP-4 format 2-21
 COMPUTATIONAL-5 or COMP-5 format 2-27,
 6-39
 COMPUTATIONAL-6 or COMP-6 A-7
 COMPUTATIONAL, COMP, BINARY
 format 2-21
 COMPUTE statement 7-29
 computer memory natural boundaries 2-22
 concepts
 algebraic signs 2-19
 COMPUTATIONAL-X or COMP-X
 format 2-28
 COMPUTATIONAL-3, COMP-3, or
 PACKED-DECIMAL format 2-25
 COMPUTATIONAL-4 or COMP-4 2-21
 COMPUTATIONAL-5 or COMP-5
 format 2-27
 COMPUTATIONAL, COMP, BINARY 2-21
 explicit and implicit attributes 2-35
 explicit and implicit Procedure Division ref-
 erences 2-34
 explicit and implicit scope terminators 2-36
 explicit and implicit specifications 2-34
 explicit and implicit transfers of
 control 2-34
 item alignment 2-20
 of classes of data 2-18
 of computer-independent data
 description 2-16
 of levels 2-16
 optional division, section, and paragraph
 headings 2-37
 program structure 2-37
 selection of character representation and
 radix 2-20
 standard alignment rules 2-19
 uniqueness of reference 2-28
 condition evaluation rules 7-18
 condition name 2-7, 2-33
 condition-name condition 7-14
 condition-name rules 6-43
 conditional expressions 7-9
 conditional sentence 2-46
 conditional statement 2-46
 Configuration Section 5-4
 CONSOLE function name 5-14
 CONSOLE IS CRT clause 18-10
 constant names 2-13, 6-43

continuation of lines 2-53
 CONTINUE statement 7-31
 CONTROL clause 14-20
 conventions
 for condition names, data names, file names,
 record names and report names 11-9
 for index names 11-10
 for program names 11-9
 COPY INDEXED statement B-8
 COPY statement 9-5
 CORRESPONDING phrase 7-20, 7-66
 CRT STATUS clause 18-12
 CSP function name 5-14
 CURRENT-DATE Special Register 2-13
 CURSOR IS clause 18-11
 C01 through C12 function names 5-14

D

data description entry in the interprogram communication module 11-25
 Data Division 1-7
 AUTO clause 18-21
 BACKGROUND-COLOR clause 18-22
 BELL clause 18-24
 BLANK clause 18-25
 BLANK WHEN ZERO clause 6-11, 18-26
 BLINK clause 18-27
 BLOCK CONTAINS clause 8-39
 CODE clause 14-19
 CODE-SET clause 8-40
 COLUMN clause 18-28
 COLUMN NUMBER clause 14-45
 communication description 15-4
 COMMUNICATION SECTION 15-4
 CONTROL clause 14-20
 data description 6-7
 Data General syntax supplement B-6
 DATA RECORDS clause 8-42, 13-12
 Data-name clause 14-46
 Data-name or FILLER clause 6-12
 default sign representation A-6
 entries 2-54
 file description 8-35
 file description entry 14-11
 FILE SECTION 8-34
 for file input and output 8-34
 FOREGROUND-COLOR clause 18-30
 FULL clause 18-32
 general format 2-42
 GRID clause 18-34
 GROUP INDICATE clause 14-47
 HIGHLIGHT clause 18-35
 in the communication module 15-4
 in the interprogram communication module 11-17
 in the nucleus 6-1

Data Division (*continued*)
 in the report writer module 14-11
 in the screen-handling module 18-14
 in the Sort-Merge module 13-10
 in the table-handling module 12-5
 initial valuestop 6-7
 JUSTIFIED clause 6-13, 18-36
 LABEL RECORDS 8-43
 LEFTLINE clause 18-37
 length of nonnumeric literals A-6
 level number 6-15
 level-number 14-48
 LINAGE clause 8-44
 LINE clause 18-38
 LINE NUMBER clause 14-49
 Microsoft syntax supplement
 differences between Microsoft and AIX VS
 COBOL C-10
 problem determination C-10
 NEXT GROUP clause 14-51
 noncontiguous working storage 6-6
 OCCURS clause 12-5, 18-40
 organization 2-41
 OVERLINE clause 18-42
 PAGE clause 14-22
 PICTURE clause 6-18, 18-43
 presentation rules tables 14-32
 PROMPT clause 18-45
 RECORD clause 8-47
 record description structure 6-7, 8-34
 RECORDING MODE clause 8-50
 REDEFINES clause 6-29
 RENAMES clause 6-32
 REPORT clause 14-14
 report description entry 14-16, 14-17
 report group description entry 14-16, 14-28
 REPORT SECTION 14-6, 14-16
 report structure 14-6
 REQUIRED clause 18-46
 REVERSE-VIDEO clause 18-47
 Ryan-McFarland syntax supplement A-6
 screen description — complete entry
 skeleton 18-15
 SCREEN SECTION 18-14, B-6
 SECURE clause 18-48
 SIGN clause 6-35, 14-53, 18-49
 SIZE clause 18-50
 SORT-MERGE file description 13-10
 SOURCE clause 14-55
 SUM clause 14-56
 SYNCHRONIZED clause 6-37
 TYPE clause 14-59
 UNDERLINE clause 18-51
 USAGE clause 6-39, 14-63, A-7
 USAGE IS INDEX clause 12-12
 VALUE clause 6-41, 14-64, 18-52, B-6
 VALUE OF clause 8-51
 working-storage records 6-6

Data Division (*continued*)
 WORKING-STORAGE SECTION 6-6
 ZERO-FILL clause 18-53

Data General syntax supplement
 Data Division
 SCREEN SECTION B-6
 VALUE clause B-6

dialect controlling directive B-3

Environment Division
 alternate keys B-5
 DATA SIZE clause B-4
 duplicate alternate keys B-5
 I-O control entry B-5
 INDEX SIZE clause B-4
 switch names B-4

long user-defined names B-3

Procedure Division
 ACCEPT statement B-7
 CALL statement B-8
 COPY INDEXED statement B-8
 DISPLAY statement B-8
 file sharing syntax B-8
 OPEN statement B-8
 READ statement B-8

data manipulation 14-7

DATA RECORDS clause 8-42, 13-12

data rules
 alphabetic 6-19
 alphanumeric 6-19
 alphanumeric-edited 6-19
 numeric 6-19
 numeric-edited 6-20

DATA SIZE clause B-4

Data-name 14-46

Data-name or FILLER clause 6-12

DATE 7-22

DATE-COMPILED paragraph 4-7

DAY 7-22

DAY-OF-WEEK 7-22

DBCS
See double-byte character set (DBCS)

debug module and interactive debugging
 COBOL run-time switch 17-4
 compile-time switch 17-4

Environment Division
 WITH DEBUGGING MODE clause 17-5

introduction 17-4

lines 17-13

Procedure Division
 READY TRACE statement 17-6
 RESET TRACE statement 17-7
 standard ANSI COBOL 17-4

DECLARATIVES
 description 2-44
 format 2-45

default sign representation A-6

DELETE statement 8-59

delimited scope statements 2-48

DG dialect controlling directive B-3

DISABLE statement 15-21

DISPLAY format 2-21

DISPLAY statement 7-32, 18-61, A-11, B-8

DISPLAY-1 USAGE 6-39

DIVIDE statement 7-34

divisions of a program 2-37

double byte character support
 in INITIALIZE statement 7-55

double-byte character set (DBCS)
 character-strings 2-6
 class and category 2-18
 elementary move rules 7-67
 figurative constant values 2-12
 literals 2-11
 PICTURE clause 6-20
 support 1-5
 use with relational operators 7-11
 using in comments 4-5

duplicate alternate keys B-5

E

EBCDIC 5-8

editing
 fixed insertion 6-24
 floating insertion 6-25
 rules 6-22
 simple insertion 6-23
 special insertion 6-24
 zero suppression 6-26

EJECT statement 10-5

elementary item size 6-20

elements 1-8

ENABLE statement 15-23

end program header 3-6, 11-13

ENTER statement 7-38

ENTRY statement 11-47

Environment Division 1-7
 alternate keys B-5
 ASSIGN clause A-3
 Configuration Section 5-4
 CONSOLE IS CRT clause 18-10
 CRT STATUS clause 18-12
 CURSOR IS clause 18-11
 Data General syntax supplement B-4
 DATA SIZE clause B-4
 duplicate alternate keys B-5
 file-control entry 8-18, 13-6
 FILE-CONTROL paragraph 8-17, 13-5, 14-10
 for file input and output 8-16
 general description 2-39, 5-4
 general format 2-40
 I-O control 8-29
 I-O control entry B-5
 I-O-CONTROL paragraph 13-8, 14-10

Environment Division (*continued*)
 in COBOL debug 17-5
 in the nucleus 5-1
 in the report writer module 14-10
 in the screen-handling module 18-8
 in the sort-merge module 13-4
 INDEX SIZE clause B-4
 INPUT-OUTPUT SECTION 14-10
 Microsoft syntax supplement
 differences between Microsoft and AIX VS
 COBOL C-4
 problem determination C-4, C-8
 OBJECT-COMPUTER paragraph 5-6
 organization 2-39
 ORGANIZATION clause A-5
 Ryan-McFarland syntax supplement A-3
 SOURCE-COMPUTER paragraph 5-5
 SPECIAL-NAMES paragraph 5-8, 18-9
 structure 2-39
 switch names B-4
 WITH DEBUGGING MODE clause 17-5
 EVALUATE statement 7-39
 EXAMINE statement 7-43
 EXECUTE statement 7-45
 EXHIBIT statement 7-46
 EXIT PROGRAM statement 11-50, A-8
 EXIT statement 7-48
 explicit and implicit
 attributes 2-35
 Procedure Division references 2-34
 scope terminators 2-36
 specifications 2-34
 EXTERNAL clause 11-30
 external objects
 common and initial programs 11-7
 description 11-7
 scope of names 11-8
 sharing data 11-8
 sharing files 11-8

F

FD 8-35
 figurative constant values 2-11
 figurative constants 3-4
 file connector 11-6
 file description entry in the interprogram communication module 11-19
 file I-O status codes 8-8, A-11
 file input and output
 AT END condition 8-11
 Data Division
 BLOCK CONTAINS clause 8-39
 CODE-SET clause 8-40
 DATA RECORDS clause 8-42
 file description 8-35
 FILE SECTION 8-34
 LABEL RECORDS 8-43

file input and output (*continued*)
 Data Division (*continued*)
 LINAGE clause 8-44
 RECORD clause 8-47
 record description structure 8-34
 RECORDING MODE clause 8-50
 VALUE OF clause 8-51
 Environment Division
 file-control entry 8-18
 FILE-CONTROL paragraph 8-17
 I-O control 8-29
 input-output section 8-16
 indexed 8-7
 introduction 8-6
 INVALID KEY condition 8-11
 linage-counter 8-12
 organization of indexed files 8-7
 organization of relative files 8-6
 organization of sequential files 8-6
 Procedure Division
 CLOSE statement 8-53
 COMMIT statement 8-58
 DELETE statement 8-59
 OPEN statement 8-62
 READ statement 8-68
 REWRITE statement 8-75
 START statement 8-79
 UNLOCK statement 8-85
 USE statement 8-86
 WRITE statement 8-89
 relative input-output 8-6
 sequential input-output 8-6
 sharing files 8-12
 file modes 8-12
 file name on disk B-4
 file position indicator 8-7
 file sharing syntax B-8
 file sharing. 8-12
 file-control entry 8-18, 13-6
 FILE-CONTROL paragraph 8-17, 13-5, 14-10
 fixed insertion editing 6-24
 floating insertion editing 6-25
 FOREGROUND-COLOR clause 18-30
 FORMFEED function name 5-14
 FULL clause 18-32
 function-name reference 5-13

G

G symbol in PICTURE clause 6-21
 general format 1-7
 general rules 1-8
 GENERATE statement 14-68
 GLOBAL clause 11-31
 global names 11-6
 GO TO statement 7-50

GOBACK statement 11-51
GRID clause 18-34
GROUP INDICATE clause 14-47

H

hexadecimal literals 2-10, 2-11
HIGHLIGHT clause 18-35
horizontal spacing 14-7

I

I-O control 8-29
I-O control entry B-5
I-O status 8-8
I-O-CONTROL paragraph 13-8, 14-10
Identification Division 1-7
 DATE COMPILED paragraph 4-7
 general description 2-38
 general format 2-38
 in the interprogram communication
 module 11-15
 nucleus 4-1
 organization 2-38
 PROGRAM-ID paragraph 4-6
 PROGRAM-ID paragraph and nested source
 programs 11-15
 REMARKS paragraph 4-8
 structure 2-38
identifier 2-33
IF statement 7-52
imperative sentence 2-48
imperative statement 2-47
implicit FILLER or padding bytes 2-22
implicit specifications 2-34
implicit synchronization 2-23
incompatible data 7-21
independent segments 16-5
INDEX SIZE clause B-4
indexed I-O 8-6
indexed input-output 8-7
indexing 2-31, 12-9
INDEX, USAGE IS 12-12
indicator area 1-8
initial programs 11-7
INITIALIZE statement 7-54
INITIATE statement 14-71
Input-Output Section 8-16
INSPECT statement 7-57
interactive debugging
 See debug module and interactive debugging
interprogram communication
 data description entry 11-25
 EXTERNAL clause 11-30
 file description entry 11-19
 GLOBAL clause 11-31

interprogram communication (*continued*)
 Identification Division
 PROGRAM-ID paragraph and nested
 source programs 11-15
 language concepts 11-6
 linkage section 11-17
 nested source programs 11-10
 Procedure Division
 CALL statement 11-36
 CANCEL statement 11-43
 CHAIN statement 11-45
 ENTRY statement 11-47
 EXIT PROGRAM statement 11-50
 GOBACK statement 11-51
 header 11-33
 USE BEFORE REPORTING
 statement 11-53
 USE statement 11-52
 report description entry 11-28
INVALID KEY condition 8-9, 8-11

J

JUSTIFIED clause 6-13, 18-36

K

KEPT LOCK phrase 8-68

L

LABEL RECORDS clause 8-43
language concepts
 ADDRESS 2-15
 character set 2-4
 character-strings 2-6
 CURRENT-DATE 2-13
 file connector 11-6
 global and local names 11-6
 nested source programs 11-6
 PICTURE character-strings 2-15
 report file 14-8
 RETURN-CODE 2-14
 separators 2-5
 SORT-CONTROL 2-15
 SORT-CORE-SIZE 2-15
 SORT-FILE-SIZE 2-15
 SORT-MODE-SIZE 2-15
 SORT-RETURN 2-15
 special register
 LINE-COUNTER 14-9
 PAGE-COUNTER 14-9
 PRINT-SWITCH 14-9
 structure 2-5
 subscripting 14-9

language concepts (*continued*)
 TALLY 2-13
 TIME-OF-DAY 2-14
 WHEN-COMPILED 2-14, 2-15
 language structure 2-5
 LEFTLINE clause 18-37
 length of nonnumeric literals A-6
 LENGTH-CHECK clause 18-32
 level number 6-15
 level numbers 2-16
 level-number 14-48
 library
 COPY statement 9-5
 REPLACE statement 9-10
 LINAGE clause 8-44
 LINAGE-COUNTER 8-12
 LINE clause 18-38
 LINE NUMBER clause 14-49
 LINE NUMBER clause notation 14-34
 LINE NUMBER clause sequence
 substitutions 14-34
 LINKAGE SECTION 11-17
 listing control
 EJECT statement 10-5
 SKIP1, SKIP2, SKIP3 statements 10-4
 TITLE statement 10-6
 literals 2-9
 literals as CALL parameters A-8
 local names 11-6
 locked records 8-68, A-13
 logic error condition 8-10

M

manual formats
 areas A and B 1-9
 elements 1-8
 general rules 1-8
 indicator area 1-8
 sequence number 1-8
 source 1-8
 syntax rules 1-8
 memory natural boundaries 2-20, 2-22
 MERGE statement 13-14, 16-8
 Message Control System (MCS) 15-4
 Microsoft syntax supplement
 compatibility of AIX VS COBOL with Micro-
 soft COBOL C-3
 dialect controlling directives C-3
 problem determination
 Data Division C-10
 documentation differences C-13
 Environment Division C-9
 extension subroutines C-13
 file input and output C-11
 general C-9
 Procedure Division C-10
 screen-handling C-12

Microsoft syntax supplement (*continued*)
 summary of syntactic differences
 Data Division C-4
 Environment Division C-4
 Procedure Division C-5
 special registers LIN and COL C-3
 MOVE statement 7-65
 multiple record locks 8-14
 MULTIPLE REEL 8-25
 multiple results in arithmetic statements 7-21
 MULTIPLE UNIT 8-25
 MULTIPLY statement 7-69

N

Native 5-8
 negated simple condition 7-16
 nested source programs 11-6
 END PROGRAM header 11-13
 initial state of a program 11-12
 organization 11-11
 structure 11-11
 NEXT GROUP clause 14-51
 NEXT special register 6-43
 noncontiguous working storage 6-6
 nonnumeric literals 2-10
 NOT ON SIZE ERROR phrase 7-19
 nucleus
 Data Division
 BLANK WHEN ZERO clause 6-11
 data description 6-7
 Data-name or FILLER clause 6-12
 initial valuestop 6-7
 JUSTIFIED clause 6-13
 level number 6-15
 noncontiguous working storage 6-6
 PICTURE clause 6-18
 record description structure 6-7
 REDEFINES clause 6-29
 RENAMES clause 6-32
 SIGN clause 6-35
 SYNCHRONIZED clause 6-37
 USAGE clause 6-39
 VALUE clause 6-41
 working-storage records 6-6
 end program header 3-6
 Environment Division
 Configuration Section 5-4
 general description 5-4
 OBJECT-COMPUTER paragraph 5-6
 SOURCE-COMPUTER paragraph 5-5
 SPECIAL-NAMES paragraph 5-8
 figurative constants 3-4
 function 3-4
 Identification Division
 DATE COMPILED paragraph 4-7
 general description 4-4
 PROGRAM-ID paragraph 4-6

nucleus (*continued*)

- Identification Division (*continued*)
 - REMARKS paragraph 4-8
- name characteristics 3-4
- overall language 3-4
- Procedure Division
 - abbreviated combined relation
 - conditions 7-17
 - ACCEPT statement 7-22
 - ADD statement 7-24
 - ALTER statement 7-27
 - arithmetic expressions 7-7
 - arithmetic operators 7-8
 - arithmetic statement rules 7-20
 - common phrases 7-19
 - complex conditions 7-15
 - COMPUTE statement 7-29
 - condition evaluation rules 7-18
 - conditional expressions 7-9
 - CONTINUE statement 7-31
 - CORRESPONDING phrase 7-20
 - DISPLAY statement 7-32
 - DIVIDE statement 7-34
 - ENTER statement 7-38
 - EVALUATE statement 7-39
 - EXAMINE statement 7-43
 - EXECUTE statement 7-45
 - EXHIBIT statement 7-46
 - EXIT statement 7-48
 - general format 7-6
 - GO TO statement 7-50
 - IF statement 7-52
 - incompatible data rule 7-21
 - INITIALIZE statement 7-54
 - INSPECT statement 7-57
 - MOVE statement 7-65
 - multiple results in arithmetic statement
 - rules 7-21
 - MULTIPLY statement 7-69
 - negated simple condition 7-16
 - NOT ON SIZE ERROR phrase 7-19
 - ON SIZE ERROR phrase 7-19
 - ON statement 7-71
 - overlapping operand rules 7-21
 - PERFORM statement 7-73
 - ROUNDED phrase 7-19
 - SET statement 7-84
 - signed receiving item rule 7-21
 - STOP statement 7-86
 - STRING statement 7-87
 - SUBTRACT statement 7-91
 - TRANSFORM statement 7-94
 - UNSTRING statement 7-96
- reference format 3-4
- subscripting 3-4
- numeric data rules 6-19
- numeric literals 2-10

numeric-edited data rules 6-20

O

- OBJECT-COMPUTER paragraph 5-6
- OCCURS clause 12-5, 18-40
- ON SIZE ERROR phrase 7-19
- ON statement 7-71
- OPEN statement 8-62, 14-72, B-8
- optional words 2-9
- organization
 - indexed files 8-7
 - sequential files 8-6
- ORGANIZATION clause 8-19, A-5
- overlapping operands 7-21, 12-13
- OVERLINE clause 18-42

P

- PACKED-DECIMAL format 2-25
- PAGE clause 14-22
- page footing presentation rules 14-41
- page heading group presentation rules 14-36
- page regions 14-25
- paragraph 1-7
- paragraphs
 - DATE COMPILED 4-7
 - description 2-44
 - FILE-CONTROL 8-17, 13-5, 14-10
 - header and name 2-54
 - I-O control 8-29
 - I-O-CONTROL 13-8, 14-10
 - OBJECT-COMPUTER 5-6
 - PROGRAM ID 4-6
 - PROGRAM-ID 11-15
 - reference format 2-54
 - REMARKS 4-8
 - SOURCE-COMPUTER 5-5
 - SPECIAL-NAMES 5-8, 18-9
- PERFORM statement 7-73, 16-8, A-9
- permanent error condition 8-10
- PICTURE character-strings 2-15
- PICTURE clause 6-18, 18-43
- POINTER
 - clause 6-40, 6-44
 - format 2-28
 - usage 7-13, 7-67, 7-85, 11-18, 11-33, 11-40
- Pointer usage 7-20
- presentation rules
 - body group 14-38
 - page footing 14-41
 - page heading group 14-36
 - report footing 14-43
 - report heading group 14-34
 - tables 14-32

presentation rules tables
 body group presentation rules 14-38
 LINE NUMBER clause notation 14-34
 LINE NUMBER clause sequence substitutions 14-34
 organization 14-33
 page footing presentation rules 14-41
 page heading group presentation rules 14-36
 report footing presentation rule 14-43
 report heading group presentation rules 14-34
 saved next group integer description 14-34
 PRINT-SWITCH 14-9, 14-73
 Procedure Division 1-7
 abbreviated combined relation conditions 7-17
 ACCEPT MESSAGE COUNT statement 15-20
 ACCEPT statement 7-22, 18-55, A-10, B-7
 ADD statement 7-24
 ALTER statement 7-27
 arithmetic expressions 7-7
 arithmetic operators 7-8
 arithmetic statement rules 7-20
 body 2-45
 bound checking A-8
 CALL statement 11-36, B-8
 CANCEL statement 11-43
 CHAIN statement 11-45
 CLOSE statement 8-53, 14-67
 combined and negated combined condition 7-16
 COMMIT statement 8-58
 common phrases 7-19
 COMPUTE statement 7-29
 condition evaluation rules 7-18
 conditional expressions 7-9
 CONTINUE statement 7-31
 COPY INDEXED statement B-8
 CORRESPONDING phrase 7-20
 Data General syntax supplement B-7
 declaratives 2-44
 DELETE statement 8-59
 DISABLE statement 15-21
 DISPLAY statement 7-32, 18-61, A-11, B-8
 DIVIDE statement 7-34
 ENABLE statement 15-23
 ENTER statement 7-38
 ENTRY statement 11-47
 EVALUATE statement 7-39
 EXAMINE statement 7-43
 EXECUTE statement 7-45
 execution 2-44
 EXHIBIT statement 7-46
 EXIT PROGRAM statement 11-50, A-8
 EXIT statement 7-48
 file I-O status codes A-11
 file sharing syntax B-8
 Procedure Division (*continued*)
 for file input and output 8-53
 general format 2-45, 7-6
 GENERATE statement 14-68
 GO TO statement 7-50
 GOBACK statement 11-51
 header 2-45, 11-33
 IF statement 7-52
 in COBOL debug 17-6
 in the communication module 15-19
 in the interprogram communication module 11-33
 in the nucleus 7-1
 in the report writer module 14-66
 in the screen-handling module 18-54
 in the sort-merge module 13-14
 in the table-handling module 12-13
 incompatible data 7-21
 INITIALIZE statement 7-54
 INITIATE statement 14-71
 INSPECT statement 7-57
 literals as CALL parameters A-8
 locked records A-13
 MERGE statement 13-14
 Microsoft syntax supplement
 differences between Microsoft and AIX VS COBOL C-5
 problem determination C-5
 MOVE statement 7-65
 multiple results in arithmetic statements 7-21
 MULTIPLY statement 7-69
 negated simple condition 7-16
 nonstandard operations on alphanumeric data items A-9
 NOT ON SIZE ERROR phrase 7-19
 ON SIZE ERROR phrase 7-19
 ON statement 7-71
 OPEN statement 8-62, 14-72, B-8
 overlapping operands 7-21, 12-13
 paragraph 2-44
 PERFORM statement 7-73, A-9
 procedure names A-9
 procedures 2-44
 PURGE statement 15-25
 READ statement 8-68, B-8
 READY TRACE statement 17-6
 RECEIVE statement 15-26
 RECORD CONTAINS clause 13-13
 RELEASE statement 13-18
 report writer statements 14-8
 RESET TRACE statement 17-7
 RETURN statement 13-19
 REWRITE on line-sequential files A-9
 REWRITE statement 8-75
 ROUNDED phrase 7-19
 Ryan-McFarland syntax supplement A-8
 SEARCH statement 12-14

Procedure Division (*continued*)

section 2-44
SEND statement 15-29
SET statement 7-84, 12-19
signed receiving items 7-21
size allocation for index data items A-8
SORT statement 13-21
START statement 8-79
statements and sentences 2-46
STOP RUN statement A-8
STOP statement 7-86
STRING statement 7-87
SUBTRACT statement 7-91
SUPPRESS statement 14-73
TERMINATE statement 14-74
TRANSFORM statement 7-94
UNLOCK statement 8-85
UNSTRING statement 7-96
USE BEFORE REPORTING
statement 11-53, 14-76
USE statement 8-86, 11-52
USING phrase 11-33
WRITE statement 8-89
procedure names A-9
program flow restrictions 16-7
program segments 16-4, 16-6
program structure 1-7, 2-37
PROGRAM-ID paragraph 4-6, 11-15
PROMPT clause 18-45
pseudo-text 2-53
PURGE statement 15-25

Q

qualification 2-29

R

READ statement 8-68, B-8
READY TRACE statement 17-6
RECEIVE statement 15-26
RECORD clause 8-47
RECORD CONTAINS clause 13-13
record description structure 6-7
record locking
multiple 8-14
single 8-13
RECORDING MODE clause 8-50
REDEFINES clause 6-29
reel devices 8-6
reference format 2-52
reference modification 2-32
relative I-O 8-6
RELEASE statement 13-18
REMARKS paragraph 4-8

RENAMES clause 6-32
REPLACE statement 9-10
REPORT clause 14-14
REPORT description entry 14-17
report description entry in the interprogram
communication module 11-28
report footing presentation rules 14-43
report group description entry 14-28
report heading group presentation rules 14-34
report writer module

Data Division

CODE clause 14-19
COLUMN NUMBER clause 14-45
CONTROL clause 14-20
Data-name clause 14-46
GROUP INDICATE clause 14-47
level-number 14-48
LINE NUMBER clause 14-49
NEXT GROUP clause 14-51
PAGE clause 14-22
REPORT clause 14-14
report description entry 14-16, 14-17
report group description entry 14-16,
14-28
REPORT SECTION 14-16
SIGN clause 14-53
SOURCE clause 14-55
SUM clause 14-56
TYPE clause 14-59
USAGE clause 14-63
VALUE clause 14-64

Environment Division

FILE-CONTROL paragraph 14-10
I-O-CONTROL paragraph 14-10
file description entry 14-11
INPUT-OUTPUT SECTION 14-10
language concepts 14-8

Procedure Division

CLOSE statement 14-67
GENERATE statement 14-68
INITIATE statement 14-71
OPEN statement 14-72
SUPPRESS statement 14-73
TERMINATE statement 14-74
USE BEFORE REPORTING
statement 14-76
section 14-6
structure 14-6
subdivisions 14-7
report writer statements 14-8
REQUIRED clause 18-46
reserved word list D-3
reserved words 2-8
RESET TRACE statement 17-7
RETURN statement 13-19
RETURN-CODE Special Register 2-14
REVERSE-VIDEO clause 18-47

REWIND 8-25
 REWRITE on line-sequential files A-9
 REWRITE statement 8-75
 ROUNDED phrase 7-19
 run time environment error 8-11
 Ryan-McFarland syntax supplement
 Data Division
 default sign representation A-6
 length of nonnumeric literals A-6
 USAGE clause A-7
 Environment Division
 ASSIGN clause A-3
 ORGANIZATION clause A-5
 introduction A-3
 Procedure Division
 ACCEPT statement A-10
 bound checking A-8
 DISPLAY statement A-11
 EXIT PROGRAM statement A-8
 file I-O status codes A-11
 literals as CALL parameters A-8
 locked records A-13
 nonstandard operations on alphanumeric
 data items A-9
 PERFORM statement A-9
 procedure names A-9
 REWRITE on line-sequential files A-9
 size allocation for index data items A-8
 STOP RUN statement A-8

S

SAA conformance 1-4, D-3
 SAME clause 8-32, 13-8
 saved next group integer description 14-34
 scope of names 11-8
 scope terminators 2-36
 screen handling
 Data Division
 AUTO clause 18-21
 BACKGROUND-COLOR clause 18-22
 BELL clause 18-24
 BLANK clause 18-25
 BLANK WHEN ZERO clause 18-26
 BLINK clause 18-27
 COLUMN clause 18-28
 FOREGROUND-COLOR clause 18-30
 FULL clause 18-32
 GRID clause 18-34
 HIGHLIGHT clause 18-35
 JUSTIFIED clause 18-36
 LEFTLINE clause 18-37
 LINE clause 18-38
 OCCURS clause 18-40
 OVERLINE clause 18-42
 PICTURE clause 18-43
 PROMPT clause 18-45
 REQUIRED clause 18-46

screen handling (*continued*)
 Data Division (*continued*)
 REVERSE-VIDEO clause 18-47
 screen description — complete entry skeleton 18-15
 SCREEN SECTION 18-14
 SECURE clause 18-48
 SIGN clause 18-49
 SIZE clause 18-50
 UNDERLINE clause 18-51
 VALUE clause 18-52
 ZERO-FILL clause 18-53
 Environment Division
 CONSOLE IS CRT clause 18-10
 CRT STATUS clause 18-12
 CURSOR IS NAME 18-11
 SPECIAL-NAMES paragraph 18-9
 introduction 18-6
 Procedure Division
 ACCEPT statement 18-55
 DISPLAY statement 18-61
 SEARCH statement 12-14
 section 2-44, 2-54
 SECURE clause 18-48
 SEGMENT-LIMIT 16-7
 segment-numbers 16-6
 segmentation module
 classification 16-5
 control 16-5
 fixed portion 16-4
 general description 16-4
 independent segments 16-5
 introduction 16-4
 organization 16-4
 program segments 16-4
 restrictions
 ALTER statement 16-7
 MERGE statement 16-8
 PERFORM statement 16-8
 SORT statement 16-8
 structure
 SEGMENT-LIMIT 16-7
 segment-numbers 16-6
 structure of program segments 16-6
 SELECT statement 8-19
 SEND statement 15-29
 sentence 2-44
 separators 2-5
 sequence number 1-8
 sequence numbers 2-53
 sequential I-O 8-6
 SET statement 7-84, 12-19
 sharing data 11-8
 sharing files 8-12, 11-8
 SIGN clause 6-35, 14-53, 18-49
 sign condition 7-15
 signed receiving items 7-21

simple insertion editing 6-23
 single record lock 8-13
 SIZE clause 18-50
 SKIP1, SKIP2, SKIP3 statements 10-4
 SORT statement 13-21, 16-8
 SORT STATUS 13-6
 SORT-CONTROL Special Register 2-15
 SORT-CORE-SIZE Special Register 2-15
 SORT-FILE-SIZE Special Register 2-15
 sort-merge
 Data Division
 DATA RECORDS clause 13-12
 file description 13-10
 Environment Division
 file-control entry 13-6
 FILE-CONTROL paragraph 13-5
 I-O-CONTROL paragraph 13-8
 Procedure Division
 MERGE statement 13-14
 RECORD CONTAINS clause 13-13
 RELEASE statement 13-18
 RETURN statement 13-19
 SORT statement 13-21
 relationship with file input and output 13-4
 SORT-MERGE file description 13-10
 SORT-MESSAGE Special Register 2-15
 SORT-MODE-SIZE Special Register 2-15
 SORT-RETURN Special Register 2-15
 SOURCE clause 14-55
 source format 1-8
 SOURCE-COMPUTER paragraph 5-5
 special insertion editing 6-24
 special register (MF extension)
 NEXT 6-43
 special register (report writer module)
 LINE-COUNTER 14-9
 PAGE-COUNTER 14-9
 PRINT-SWITCH 14-9
 special registers (language concepts)
 ADDRESS 2-15
 CURRENT-DATE 2-13
 RETURN-CODE 2-14
 SORT-CONTROL 2-15
 SORT-CORE-SIZE 2-15
 SORT-FILE-SIZE 2-15
 SORT-MESSAGE 2-15
 SORT-MODE-SIZE 2-15
 SORT-RETURN 2-15
 TALLY 2-13
 TIME-OF-DAY 2-14
 WHEN-COMPILED 2-14, 2-15
 SPECIAL-NAMES paragraph 5-8, 18-9
 split-key name 8-23
 standard alignment rules 2-19
 START statement 8-79
 statements
 ACCEPT 7-22, 18-55, A-10, B-7
 ACCEPT MESSAGE COUNT 15-20

statements (*continued*)
 ADD 7-24
 ALTER 7-27, 16-7
 arithmetic 7-20
 CALL 11-36, B-8
 CANCEL 11-43
 CHAIN 11-45
 CLOSE 8-53, 14-67
 COMMIT 8-58
 COMPUTE 7-29
 conditional 2-46
 CONTINUE 7-31
 COPY 9-5
 COPY INDEXED B-8
 DELETE 8-59
 delimited scope statements 2-48
 DISABLE 15-21
 DISPLAY 7-32, 18-61, A-11, B-8
 DIVIDE 7-34
 EJECT 10-5
 ENABLE 15-23
 ENTER 7-38
 ENTRY 11-47
 EVALUATE 7-39
 EXAMINE 7-43
 EXECUTE 7-45
 EXHIBIT 7-46
 EXIT 7-48
 EXIT PROGRAM 11-50, A-8
 GENERATE 14-68
 GO TO 7-50
 GOBACK 11-51
 IF 7-52
 imperative 2-47
 INITIALIZE 7-54
 INITIATE 14-71
 INSPECT 7-57
 MERGE 13-14, 16-8
 MOVE 7-65
 MULTIPLY 7-69
 ON 7-71
 OPEN 8-62, 14-72, B-8
 PERFORM 7-73, 16-8, A-9
 PURGE 15-25
 READ 8-68, B-8
 READY TRACE 17-6
 RECEIVE 15-26
 RELEASE 13-18
 REPLACE 9-10
 RESET TRACE 17-7
 RETURN 13-19
 REWRITE 8-75
 SEARCH 12-14
 SEND 15-29
 SET 7-84, 12-19
 SKIP1, SKIP2, SKIP3 10-4
 SORT 13-21, 16-8
 START 8-79

statements (*continued*)
 STOP 7-86
 STOP RUN A-8
 STRING 7-87
 SUBTRACT 7-91
 SUPPRESS 14-73
 TERMINATE 14-74
 TITLE 10-6
 TRANSFORM 7-94
 UNLOCK 8-85
 UNSTRING 7-96
 USE 8-86, 11-52
 USE BEFORE REPORTING 11-53, 14-76
 WRITE 8-89

status keys 8-8
 STOP RUN statement A-8
 STOP statement 7-86
 STRING statement 7-87
 subscripting 2-30
 SUBTRACT statement 7-91
 SUM clause 14-56
 supported language elements of IBM OS/VS
 COBOL 85 high-level 1-4
 SUPPRESS statement 14-73
 switch names B-4
 switch-status condition 7-15
 SYNCHRONIZED clause 2-22, 6-37
 synchronization
 description 2-22
 memory natural boundaries 2-20
 syntactic differences
 Environment Division C-4
 Procedure Division C-5
 special registers LIN and COL C-3
 syntax problem determination
 documentation differences C-13
 Environment Division C-9
 extension subroutines C-13
 file input and output C-11
 general C-9
 Procedure Division C-10
 screen-handling C-12
 syntax rules 1-8
 SYSIN function name 5-14
 SYSIPT function name 5-14
 SYSLIST function name 5-14
 SYSLST function name 5-14
 SYSOUT function name 5-14
 SYSPNCH function name 5-14
 SYSPUNCH function name 5-14
 S01, S02 function names 5-14

T

tab expansion 8-76
 TAB function name 5-14

table-handling
 Data Division
 OCCURS clause 12-5
 USAGE IS INDEX clause 12-12
 introduction 12-4
 Procedure Division
 overlapping operands 12-13
 SEARCH statement 12-14
 SET statement 12-19
 TALLY Special Register 2-13
 tape devices 8-6
 TERMINATE statement 14-74
 TIME 7-22
 TIME-OF-DAY Special Register 2-14
 TITLE statement 10-6
 transfers of control 2-34
 TRANSFORM statement 7-94
 truncation 2-24
 TYPE clause 14-59

U

UNDERLINE clause 18-51
 uniqueness of reference 2-28
 UNLOCK statement 8-85
 UNSTRING statement 7-96
 USAGE clause 6-39, 14-63, A-7
 USAGE IS INDEX clause 12-12
 USE BEFORE REPORTING statement 11-53, 14-76
 USE FOR DEBUGGING 17-8
 USE statement 8-86, 11-52
 user-defined names B-3
 user-defined words 2-7
 USING phrase 11-33, 11-36

V

VALUE clause 6-41, 14-64, 18-52, B-6
 VALUE OF clause 8-51
 vertical spacing 14-6
 VS COBOL
 debug run-time switch 17-4
 directing statement 2-47
 enhancements 1-5
 Identification Division 2-38
 language elements 1-4
 manual format 1-7
 program structure 1-7, 2-37
 source program 3-5
 standard debug 17-4
 supported elements 1-4
 words 2-6

W

WHEN-COMPILED Special Register 2-14
WITH DEBUGGING MODE clause 17-5
word storage mode 2-22
working-storage records 6-6
WORKING-STORAGE SECTION 6-6
WRITE statement 8-89

Z

zero suppression editing 6-26
ZERO-FILL clause 18-53

Reader's Comment Form

Language Reference for IBM AIX VS COBOL Compiler/6000

SC23-2177-00

Please use this form only to identify publication errors or to request changes in publications. Your comments assist us in improving our publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your IBM-approved remarketer. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

- If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.
- If you would like a reply, check this box. Be sure to print your name and address below.

Page	Comments

Please contact your IBM representative or your IBM-approved remarketer to request additional publications.

Please print

Date _____

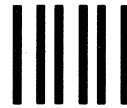
Your Name _____

Company Name _____

Mailing Address _____

Phone No. () _____
Area Code

No postage necessary if mailed in the U.S.A



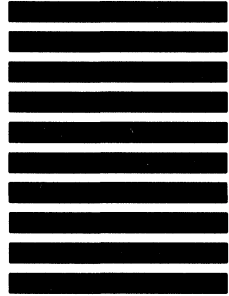
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 997
11400 Burnet Rd.
Austin, Texas 78758-3493



Fold

Fold

Cut or Fold Along Line

Fold and Tape

Please Do Not Staple

Fold and Tape



© IBM Corp. 1990

International Business Machines
Corporation
11400 Burnet Road
Austin, Texas 78758-3493

Printed in the
United States of America
All Rights Reserved

SC23-2177-00

SC23-2177-00

