LIANT

Retooling Enterprise Systems

RM/COBOL®

Language Reference Manual

Version 7.5

This document contains the information required to develop COBOL language programs using the Liant Software Corporation RM/COBOL compiler. This document contains little tutorial material; nevertheless, it should be of value to the novice as well as the experienced programmer.

For operating system dependent information, the reader should refer to the appropriate RM/COBOL user's guide.

The information in this document is subject to change without prior notice. Liant Software Corporation assumes no responsibility for any errors that may appear in this document. Liant reserves the right to make improvements and/or changes in the products and programs described in this manual at any time without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopied, recorded, or otherwise, without prior written permission of Liant Software Corporation.

The software described in this document is furnished to the user under a license for a specific number of uses and may be copied (with inclusion of the copyright notice) only in accordance with the terms of such license.

Copyright © 1985, 1986–2002 by Liant Software Corporation. All rights reserved. Printed in the U.S.A.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein

> FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC ® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translater Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

RM, RM/COBOL, RM/COBOL-85, Relativity, Enterprise CodeBench, RM/InfoExpress, RM/Panels, VanGui Interface Builder, CodeWatch, CodeBridge, Cobol-WOW, InstantSQL, Liant, and the Liant logo are trademarks or registered trademarks of Liant Software Corporation.
IBM and Macro Assembler/2 are trademarks or registered trademarks of International Business Machines Corporation.
Novell and NetWare are trademarks or registered trademarks of Novell, Incorporated.
UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Ltd.
All other products, brand, or trade names used in this publication are the trademarks or registered trademarks of their respective trademark holders, and are used only for explanation purposes.

Document Number 401206-0502

Table of Contents

Preface xix
Organization of Information xix
Conventions and Symbols xx
Related Publications xxi

Chapter 1: Language Structure 1-1
Character Set 1-1
Separators 1-1
Character-Strings 1-4
COBOL Words 1-4
User-Defined Words 1-4
System-Names 1-8
Reserved Words 1-10
Context-Sensitive Words 1-14
Literals 1-14
Numeric Literals 1-14
Nonnumeric Literals 1-14
Figurative Constants 1-15
PICTURE Character-Strings 1-17
Comment-Entry 1-17
Program Structure 1-18
Source Format 1-18
Continuation of Lines 1-19
Blank Lines 1-19
Comment Lines 1-19
In-Line Comments 1-20
Debugging Lines 1-20
Statements 1-21
Directive Statements 1-21
Conditional Statements 1-21
Conditional Phrases 1-22
Imperative Statements 1-22
Delimited Scope Statements 1-23
Scope of Statements 1-23
Sentences 1-24
Clauses and Entries 1-24
Paragraphs 1-24
Sections 1-24

Divisions ... 1-25
Source Program General Format ... 1-25
Inter-Program Communication ... 1-26
Nested Source Programs ... 1-26
File Connector ... 1-27
Global Names and Local Names ... 1-27
External Objects and Internal Objects ... 1-28
Common Programs and Initial Programs ... 1-29
Sharing Data in a Run Unit ... 1-29
Sharing Files in a Run Unit ... 1-30
Scope of Names ... 1-30
Program-Names ... 1-31
Condition-Names, Constant-Names, Data-Names, File-Names, Record-Names and Split-Key-Names ... 1-32
Index-Names ... 1-33
Initial State of a Program ... 1-33
End Program Header ... 1-34
COPY Statement ... 1-35
REPLACE Statement ... 1-40

Chapter 2: Identification Division ... 2-1
Identification Division Structure ... 2-1
Program Identification ... 2-2
PROGRAM-ID Paragraph ... 2-2
AUTHOR, INSTALLATION, DATE-WRITTEN, SECURITY, and REMARKS Paragraphs ... 2-3
DATE-COMPILED Paragraph ... 2-3

Chapter 3: Environment Division ... 3-1
Environment Division Structure ... 3-1
Configuration Section ... 3-5
SOURCE-COMPUTER Paragraph ... 3-5
OBJECT-COMPUTER Paragraph ... 3-6
SPECIAL-NAMES Paragraph ... 3-7
Mnemonic-Name Clause ... 3-9
ALPHABET Clause ... 3-10
Code Name Alphabets ... 3-13
Literal Alphabets ... 3-13
Indexed File Alphabets ... 3-15
EBCDIC Translation ... 3-15
SYMBOLIC CHARACTERS Clause ... 3-15
CLASS Clause ... 3-16

CURRENCY SIGN Clause 3-17
DECIMAL-POINT Clause 3-18
Input-Output Section 3-18
FILE-CONTROL Paragraph 3-19
File Control Entry 3-20
SELECT Clause 3-21
ACCESS MODE Clause 3-23
ASSIGN Clause 3-25
CODE-SET Clause 3-26
COLLATING SEQUENCE Clause 3-27
FILE STATUS Clause 3-28
LOCK MODE Clause 3-28
ORGANIZATION Clause 3-29
Sequential 3-29
Relative 3-29
Indexed 3-30
PADDING CHARACTER Clause 3-30
RECORD DELIMITER Clause 3-31
RECORD KEY and ALTERNATE RECORD KEY Clauses 3-32
RESERVE Clause 3-34
Sort-Merge File Control Entry 3-35
SELECT Clause 3-35
ASSIGN Clause 3-35
I-O-CONTROL Paragraph 3-36
RERUN Clause 3-36
SAME Clause 3-38
MULTIPLE FILE Clause 3-40

Chapter 4: Data Division 4-1
Data Division Structure 4-1
File Section 4-4
File Description Entry 4-5
Sort-Merge File Description Entry 4-6
File Description Clauses 4-7
BLOCK CONTAINS Clause 4-7
CODE-SET Clause 4-8
DATA RECORDS Clause 4-8
EXTERNAL Clause 4-9
GLOBAL Clause 4-9
LABEL RECORDS Clause 4-9
LINAGE Clause 4-10
RECORD Clause 4-15
VALUE OF Clause 4-17

Working-Storage Section 4-18
Linkage Section 4-19
Communication Section 4-22
Screen Section 4-22
Record Description Entry 4-23
Level-Numbers 4-23
Elementary Items 4-24
77-Level Description Entry 4-24
Data Description Entry 4-24
BLANK WHEN ZERO Clause 4-28
Data-Name or FILLER Clause 4-29
EXTERNAL Clause 4-29
GLOBAL Clause 4-30
JUSTIFIED Clause 4-30
Level-Number 4-31
OCCURS Clause 4-32
PICTURE Clause 4-34
Character-String (Data Categories) 4-35
Symbols 4-36
Editing Rules 4-40
Simple Insertion Editing 4-41
Special Insertion Editing 4-41
Fixed Insertion Editing 4-41
Floating Insertion Editing 4-42
Zero Suppression Editing 4-43
PICTURE Symbol Precedence 4-44
REDEFINES Clause 4-46
RENAMES Clause 4-47
SIGN Clause 4-48
SYNCHRONIZED Clause 4-50
USAGE Clause 4-52
COMPUTATIONAL Usage 4-53
COMPUTATIONAL-1 Usage 4-54
COMPUTATIONAL-3 or PACKED-DECIMAL Usage 4-54
COMPUTATIONAL-4 or BINARY Usage 4-55
COMPUTATIONAL-6 Usage 4-56
DISPLAY Usage 4-56
INDEX Usage 4-56
POINTER Usage 4-57
VALUE Clause 4-58
Data Item Initialization Rules (Format 1 VALUE Clause) 4-60
Condition-Name Rules (Format 2 VALUE Clause) 4-61
Constant-Name Rules (Format 3 VALUE Clause) 4-62

Communication Description Entry 4-64
Input CD General Rules 4-67
Output CD General Rules 4-71
Input-Output CD General Rules 4-73
Status Key Conditions 4-76
Error Key Values 4-79
Screen Description Entry 4-80
AUTO Clause 4-84
BACKGROUND Clause 4-84
BELL Clause 4-85
BLANK LINE Clause 4-85
BLANK REMAINDER Clause 4-86
BLANK SCREEN Clause 4-86
BLANK WHEN ZERO Clause 4-86
BLINK Clause 4-87
COLUMN Clause 4-87
ERASE Clause 4-88
FOREGROUND Clause 4-88
FULL Clause 4-89
HIGHLIGHT and LOWLIGHT Clauses 4-89
JUSTIFIED Clause 4-90
LINE Clause 4-90
PICTURE Clause 4-92
REQUIRED Clause 4-93
REVERSE Clause 4-93
SECURE Clause 4-93
SIGN Clause 4-94
UNDERLINE Clause 4-94
USAGE Clause 4-94
VALUE Clause 4-95
Data Structures 4-95
Classes of Data 4-95
Standard Alignment Rules 4-96
Uniqueness of Reference 4-97
Qualification 4-97
Subscripting 4-100
Reference Modification 4-102
Identifier 4-103
Condition-Name 4-103
Table Handling 4-104
Table Definition 4-104
References to Table Items 4-106

Chapter 5: Procedure Division 5-1
Procedure Division Header 5-1
Procedure Division Structure 5-4
Procedures 5-6
Execution 5-6
Procedure References 5-7
Explicit and Implicit Transfers of Control 5-7
Segmentation 5-9
Segments 5-9
Fixed Portion 5-9
Independent Segments 5-10
Segmentation Classification 5-11
Segmentation Control 5-11
Restrictions on Program Flow 5-11
ALTER Statement Restrictions 5-11
PERFORM Statement Restrictions 5-12
MERGE Statement Restrictions 5-12
SORT Statement Restrictions 5-12
USE Statement 5-13
Common Rules 5-16
Subscript Evaluation 5-16
Arithmetic Statements 5-16
Modes of Operation 5-16
Composite Size 5-17
ROUNDED Phrase 5-17
Size Error Condition 5-18
Overlapping Operands 5-19
Incompatible Data 5-20
Arithmetic Expressions 5-20
Arithmetic Operators 5-22
Formation and Evaluation Rules 5-22
Conditional Expressions 5-23
Simple Conditions 5-23
Relation Condition 5-24
Comparison of Numeric Operands 5-25
Comparison of Nonnumeric Operands 5-26
Comparisons of Index-Names and Index Data Items 5-27
Comparison of Pointer Data Items 5-27
LIKE Condition (Special Case of Relation Condition) 5-27
Class Condition 5-37
Sign Condition 5-39
Condition-Name Condition (Conditional Variable) 5-39
Switch-Status Condition 5-40

Complex Conditions 5-40
Negated Conditions 5-41
Combined Conditions 5-41
Abbreviated Combined Relation Conditions 5-42
Condition Evaluation Rules 5-43
Sequential Organization Input-Output 5-43
Function 5-43
Organization 5-44
Access Mode 5-44
File Position Indicator 5-44
I-O Status 5-44
At End Condition 5-49
Relative Organization Input-Output 5-50
Function 5-50
Organization 5-50
Access Modes 5-51
File Position Indicator 5-51
I-O Status 5-51
Invalid Key Condition 5-56
At End Condition 5-57
Indexed Organization Input-Output 5-59
Function 5-59
Organization 5-59
Access Modes 5-60
File Position Indicator 5-60
I-O Status 5-60
Invalid Key Condition 5-65
At End Condition 5-67
File Locking 5-68
Record Locking 5-69
Record Locking Modes 5-71
Automatic Record Locking Modes 5-71
Manual Record Locking Modes 5-72
Single Record Locking Modes 5-73
Multiple Record Locking Modes 5-73
Interactive Terminal I-O 5-74
Sort-Merge 5-75
Communication Facility 5-75
Message Control System 5-75
Object Program 5-76
Relationship of the Object Program to the Message Control System and Communication Devices 5-76
Invoking the Object Program 5-77

Scheduled Initiation of the Object Program 5-78
Invocation of the Object Program by the Message Control System 5-78
Determining the Method of Scheduling 5-78
Concept of Messages and Message Segments 5-79
Concept of Queues 5-80
Independent Enqueueing and Dequeueing 5-80
Enabling and Disabling Queues 5-81
Queue Hierarchy 5-81

Chapter 6: Procedure Division Statements 6-1

ACCEPT . . . FROM Statement 6-2
ACCEPT Statement (Terminal I-O) 6-6
NO BEEP Phrase 6-9
BLINK Phrase 6-10
CONTROL Phrase 6-10
CONVERT Phrase 6-11
CURSOR Phrase 6-13
ECHO Phrase 6-13
ERASE Phrase 6-14
ON EXCEPTION and NOT ON EXCEPTION Phrases 6-14
HIGH, LOW and OFF Phrases 6-17
LINE and POSITION Phrases 6-17
Determining Line and Position 6-18
PROMPT Phrase 6-19
REVERSE Phrase 6-19
SIZE Phrase 6-19
TAB Phrase 6-20
TIME Phrase 6-21
UNIT Phrase 6-21
UPDATE Phrase 6-22
ACCEPT MESSAGE COUNT Statement 6-24
ACCEPT Screen-Name Statement 6-25
ADD Statement 6-29
CORRESPONDING Phrase 6-30
ALTER Statement 6-32
CALL Statement 6-33
USING Phrase 6-36
GIVING Phrase 6-38
OVERFLOW, EXCEPTION, and NOT EXCEPTION Phrases 6-39
CALL PROGRAM Statement 6-41
CANCEL Statement 6-43

CLOSE Statement 6-45
REEL and UNIT Phrases 6-46
NO REWIND Phrase 6-46
REMOVAL Phrase 6-47
LOCK Phrase 6-47
COMPUTE Statement 6-48
CONTINUE Statement 6-50
DELETE Statement (Relative and Indexed I-O) 6-51
DELETE FILE Statement 6-54
DISABLE Statement 6-55
INPUT Phrase 6-56
I-O TERMINAL Phrase 6-56
OUTPUT Phrase 6-56
TERMINAL Phrase 6-57
WITH KEY Phrase 6-57
DISPLAY . . . UPON Statement 6-58
DISPLAY Statement (Terminal I-O) 6-60
BEEP Phrase 6-62
BLINK Phrase 6-62
CONTROL Phrase 6-62
CONVERT Phrase 6-63
ERASE Phrase 6-64
HIGH and LOW Phrases 6-64
LINE and POSITION Phrases 6-65
Determining Line and Position 6-66
REVERSE Phrase 6-66
SIZE Phrase 6-67
UNIT Phrase 6-67
DISPLAY Screen-Name Statement 6-69
DIVIDE Statement 6-71
REMAINDER Phrase 6-73
ENABLE Statement 6-75
INPUT Phrase 6-76
I-O TERMINAL Phrase 6-76
OUTPUT Phrase 6-76
TERMINAL Phrase 6-77
WITH KEY Phrase 6-77
ENTER Statement 6-78
EVALUATE Statement 6-79
EXIT Statement 6-84
GOBACK Statement 6-87
GO TO Statement 6-88
DEPENDING ON Phrase 6-89

IF Statement 6-90
INITIALIZE Statement 6-92
INSPECT Statement 6-97
MERGE Statement 6-106
MOVE Statement 6-113
CORRESPONDING Phrase 6-116
MULTIPLY Statement 6-119
OPEN Statement 6-121
INPUT Phrase 6-124
OUTPUT Phrase 6-125
I-O Phrase 6-125
EXTEND Phrase 6-126
NO REWIND Phrase 6-126
PERFORM Statement 6-128
PURGE Statement 6-145
READ Statement 6-146
KEY Phrase 6-151
LOCK Phrase 6-152
INTO Phrase 6-153
INVALID KEY and NOT INVALID KEY Phrases 6-153
RECEIVE Statement 6-155
NO DATA and WITH DATA Phrases 6-156
MESSAGE Phrase 6-157
SEGMENT Phrase 6-157
RELEASE Statement 6-159
FROM Phrase 6-159
RETURN Statement 6-161
REWRITE Statement 6-163
FROM Phrase 6-166
SEARCH Statement 6-167
SEND Statement 6-173
ADVANCING Phrase 6-176
SET Statement 6-178
SORT Statement 6-183
START Statement (Relative and Indexed I-O) 6-191
SIZE Phrase 6-195
INVALID KEY and NOT INVALID KEY Phrases 6-195
STOP Statement 6-197
STRING Statement 6-199
DELIMITED Phrase 6-200
POINTER Phrase 6-201
OVERFLOW and NOT OVERFLOW Phrases 6-201

SUBTRACT Statement 6-203
CORRESPONDING Phrase 6-204
UNLOCK Statement 6-206
UNSTRING Statement 6-207
USE Statement 6-211
WRITE Statement 6-212
FROM Phrase 6-215
ADVANCING Phrase 6-215
END-OF-PAGE and NOT END-OF-PAGE Phrases 6-217
INVALID KEY and NOT INVALID KEY Phrases 6-218

Appendix A: Reserved Words A-1
Reserved Words A-1
Context-Sensitive Words A-7
Special Symbols A-8
Nonreserved System-Names A-9

Appendix B: Compiler Messages B-1
Compiler Messages B-1

Appendix C: Glossary C-1
Definitions C-1

Index X-1

Comments

The following last-minute changes and enhancements have been added as comments to the *RM/COBOL Language Reference Manual*:

New Compiler Feature: Implied PICTURE for Numeric VALUE 4-27

List of Figures

Figure 1-1 Source Format 1-18

Figure 4-1 General LINAGE Clause 4-14
Figure 4-2 Example LINAGE Clause 4-15

Figure 6-1 PERFORM . . . VARYING Statement 6-134
Figure 6-2 PERFORM . . . VARYING Statement 6-136
Figure 6-3 PERFORM . . . VARYING Statement 6-138
Figure 6-4 PERFORM . . . VARYING Statement 6-140
Figure 6-5 PERFORM Statement Examples 6-142
Figure 6-6 PERFORM Statement Examples 6-142
Figure 6-7 PERFORM Statement Examples 6-143
Figure 6-8 SEARCH Statement 6-171

List of Tables

Table 1-1 RM/COBOL Character Set 1-2
Table 1-2 System Names 1-9
Table 1-3 Nonnumeric Literals and Their Values 1-15
Table 1-4 Imperative Verbs 1-23

Table 4-1 PICTURE Clause Editing 4-41
Table 4-2 Editing Symbol Results 4-42
Table 4-3 Results of + and – Editing 4-43
Table 4-4 PICTURE Symbol Precedence 4-45
Table 4-5 Valid Data Item Encodings 4-49
Table 4-6 Communication Status Key Conditions 4-77
Table 4-7 Error Key Values 4-79
Table 4-8 Color Integers 4-85
Table 4-9 Interaction of LINE and COLUMN Clauses in a Screen Description Entry 4-91
Table 4-10 Data Item Relationships 4-96
Table 4-11 Example 2 Definitions 4-105

Table 5-1 Combination of Symbols in Arithmetic Expressions 5-21
Table 5-2 Arithmetic Operators 5-22
Table 5-3 Relational Operators 5-25
Table 5-4 XML Entity References 5-29
Table 5-5 Regular Expression Single-Character Escape Sequences 5-30
Table 5-6 Regular Expression Multi-Character Escape Sequences 5-31
Table 5-7 Unicode Character Property Designators 5-32
Table 5-8 Logical Operators 5-40

Table 6-1 EXCEPTION STATUS Values 6-5
Table 6-2 ACCEPT Statement Phrases and Output and Screen Fields 6-9
Table 6-3 Generic Key Names 6-16
Table 6-4 DISPLAY Statement Phrases and Output and Screen Fields 6-61
Table 6-5 Default Initialization Values 6-95
Table 6-6 Types of MOVE Statements and Their Legality 6-116
Table 6-7 Availability of a File 6-122
Table 6-8 Permissible Statements 6-123
Table 6-9 Data Item Contents 6-175
Table 6-10 SET Statement Operand Validity 6-181

Table A-1 Contest-Sensitive Words A-7
Table A-2 System-Names Used in the SPECIAL-NAMES Paragraph A-9
Table A-3 System-Names for Device Types A-10
Table A-4 System-Names for Record Delimiting Techniques A-10
Table A-5 System-Names for Labels A-11
Table A-6 System-Names for Colors A-11

Preface

RM/COBOL is a high implementation of the American National Standard COBOL X3.23-1985, designed for optimum performance and wide portability across a broad diversity of computers and operating systems. This manual provides comprehensive information about the RM/COBOL language. It provides complete syntax for all statements and detailed information on other aspects of the language.

Organization of Information

This manual is divided into the following parts:

Chapter 1—Language Structure presents detailed information on the structure of the language. This includes the structure of program units, the valid character set, words and types of statements.

Chapter 2—Identification Division details the structure and syntax of the Identification Division.

Chapter 3—Environment Division details the structure and syntax of the Environment Division.

Chapter 4—Data Division details the structure and syntax of the Data Division.

Chapter 5—Procedure Division provides general information on the Procedure Division. This includes control transfers, program segmentation and a number of other general rules. Procedure Division compiler directive statements are described in this chapter.

Chapter 6—Procedure Division Statements details the structure and syntax of all imperative and conditional statements.

Appendix A—Reserved Words lists words that are reserved, and those that are removed from the reserved word list when the RM/COBOL 2 compatibility option is selected in the Compile Command (as described Chapter 6, *Compiling*, of the *RM/COBOL User's Guide*).

Appendix B—Compiler Messages lists the informational, warning, and error messages that may be generated during compilation.

Appendix C—Glossary defines terms used to describe the RM/COBOL language.

The *RM/COBOL Language Reference Manual* also includes an index.

Conventions and Symbols

The following conventions and symbols are used or followed throughout this guide.

1. The notation for hexadecimal values is the value followed by a lowercase h (for example, 0Dh).
2. The separators comma and semicolon may be used anywhere the separator space is used in the general formats. In the source program these separators are interchangeable.
3. The separator period, when used in the formats, has the status of a required word.
4. The special character words +, –, >, <, =, >= and <=, when appearing in formats, although not underlined, are required when such portions of the formats are used.
5. The symbols found in the syntax charts are used as follows:

italicized words	indicate items for which you substitute a value
UPPERCASE WORDS	indicate optional items which—if you use them—you enter exactly as shown (although not necessarily in uppercase)
UPPERCASE WORDS	indicate required items which you enter exactly as shown (although not necessarily in uppercase)
...	indicates indefinite repetition of the last item
WORDS STACKED STACKED WORDS	indicate alternatives
[ ]	surround optional items
{ }	surround a set of alternatives, one of which is required
{| |}	surround a set of unique alternatives, one or more of which is required, but each alternative may be specified only once; when multiple alternatives are specified, they may be specified in any order.

6. In the electronic PDF file, this symbol represents a “post-it” note that allows you to view last-minute comments about a specific topic on the page in which it occurs. This same information is also contained in the readme text file under the section, Documentation Changes.

   Double-click on the note symbols to open them. You can click the note window’s Close box after you have reviewed the contents. These notes can be viewed in the Adobe Acrobat Reader but will not print, although you can copy and paste the text into another application, such as Microsoft Word, if you wish.

Related Publications

For additional information, refer to the following publications:

RM/COBOL Syntax Summary

RM/COBOL User’s Guide

CodeBridge—Calling Non-COBOL Subprograms

CodeWatch User’s Guide

Chapter 1: Language Structure

This chapter presents detailed information on the structure of the language. This includes the structure of program units, the valid character set, words and types of statements.

The smallest element in the language is the character. A character is a digit, a letter of the alphabet, punctuation or a special mark. A word is one possible result obtained when one or more characters are joined in a sequence of contiguous characters. Just as English words are determined by rules of spelling, so COBOL words are formed by following a specific set of rules.

Using syntactic and grammatical rules, words and punctuation characters are combined into statements, sentences, paragraphs and sections. When using the English language, a failure to follow the rules of grammar and sentence structure may cause misunderstanding: the same is true when writing a COBOL source program. It must be emphasized that a thorough knowledge of the rules of the language structure is a prerequisite to writing a workable program.

Character Set

The RM/COBOL character set is shown in Table 1-1. Inside nonnumeric literals and in comment-entries and comment lines, other characters may be used but have no grammatical meaning.

Characters are combined to form either a separator or a character-string.

Lowercase letters are allowed anywhere and are treated as uppercase letters except in nonnumeric literals and when used as the currency symbol in PICTURE character-strings. Within hexadecimal, nonnumeric literals, the lowercase letters a, b, c, d, e, and f are equivalent to the uppercase letters A, B, C, D, E, and F.

Separators

A separator is a string of one or more of the characters marked with a [1] in Table 1-1.

Table 1-1: RM/COBOL Character Set

Type	Representation	Name
Digits		0 through 9
Letters		A through Z a through z
Punctuation	’ : , = (. ”) ;	Apostrophe [1] Colon [1] Comma [1] Equal sign [1] Left parenthesis [1] Period [1] Quotation mark [1] Right parenthesis [1] Semicolon [1] Space [1]
Special	* $ > < – + /	Asterisk Currency Greater than Less than Minus (or hyphen) Plus Slash (or solidus)
[1] *The character can be used as a separator.*		

Separators are formed according to the following rules:

1. A space is a separator. Anywhere a space is used as a separator or as part of a separator, more than one space may be used.

2. Commas, semicolons, and periods are separators when they are immediately followed by a space. At any point in the syntax where a space is allowed, a comma separator or semicolon separator is also allowed.

3. Parentheses are separators that must appear only in balanced pairs of left and right parentheses. They delimit subscripts, reference modifiers, binary allocation values, arithmetic expressions, constant expressions, and conditions.

4. Quotation marks are separators that delimit nonnumeric literals. They must always appear in balanced pairs, except when the continuation of a nonnumeric literal is being specified.

   An opening quotation mark must be immediately preceded by a space or left parenthesis.

   A closing quotation mark must be immediately followed by a space, comma separator, semicolon separator, period separator, or right parenthesis.

   Either the quotation mark or the apostrophe may be used to delimit nonnumeric literals. The apostrophe has the same characteristics as the quotation mark, described above.

5. The punctuation character colon is a separator and is required when shown in the general formats.

6. A pair of adjacent equal signs that are not split across a continuation forms a pseudo-text delimiter. A pseudo-text delimiter is a separator.

   Pseudo-text delimiters may be used only in balanced pairs to delimit pseudo-text in COPY and REPLACE statements (see the discussions of these statements that begin on pages 1-35 and 1-40, respectively). An opening pseudo-text delimiter must be immediately preceded by a space; a closing pseudo-text delimiter must be immediately followed by one of the separators space, comma, semicolon, or period.

7. A space may immediately precede all separators except:

   a. If prohibited by specific statement syntax.

   b. If the separator is a closing quotation mark. In this case, a preceding space is considered part of the nonnumeric literal, not a separator.

   c. The opening pseudo-text delimiter, where the preceding space is required.

8. A space may immediately follow any separator except an opening quotation mark. In this case, the space is considered part of the nonnumeric literal, not a separator.

9. Any punctuation character which appears as part of the specification of a PICTURE character-string or numeric literal is not considered a punctuation character; it is treated as a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by a space, comma, semicolon or period separator. See the discussion of PICTURE character-strings on page 1-17.

These rules do not apply to characters within nonnumeric literals or comments.

Character-Strings

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

COBOL Words

A COBOL word is a character-string of not more than 30 characters which forms a user-defined word, a system-name, a context-sensitive word, or a reserved word. Each character of a COBOL word is selected from the set of letters, digits, and the hyphen. The hyphen may not appear as the first or last character. Lowercase letters are considered equivalent to the corresponding uppercase letters. Within a source program, reserved words and user-defined words form disjoint sets; reserved words and system-names form disjoint sets, system-names and user-defined words form intersecting sets.

The same COBOL word may be used as a system-name and as a user-defined word within a source program; the class of a specific occurrence of this COBOL word is determined by the context of the clause or phrase in which it occurs.

User-Defined Words

User-defined words comprise alphabetic and numeric characters, and the hyphen. A user-defined word can neither begin nor end with a hyphen. With the exception of paragraph-names, section-names, level-numbers and segment-numbers, all user-defined words must contain at least one alphabetic character.

Here are the types of user-defined words:

Alphabet-name
Cd-name
Class-name
Condition-name
Constant-name
Data-name
File-name
Index-name
Level-number
Library-name
Mnemonic-name
Paragraph-name
Program-name
Record-name
Routine-name
Screen-name
Section-name
Segment-number
Split-key-name
Symbolic-character
Text-name

Within a given source program, but excluding any contained program, the user-defined words are grouped into the following disjoint sets:

Alphabet-names
Cd-names
Class-names
Condition-names, data-names, record-names, screen-names, and split-key-names
Constant-names
File-names
Index-names
Library-names
Mnemonic-names
Paragraph-names
Program-names
Routine-names
Section-names
Symbolic-characters
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 given disjoint set must be unique, except as specified in the rules for uniqueness of reference. Segment-numbers and level-numbers need not be unique; a given specification of a segment-number or level-number may be identical to any other segment-number or level-number.

The types of user-defined words are defined as follows:

1. **Alphabet-name.** An alphabet-name identifies a character code set. It must contain at least one alphabetic character and must be unique.

2. **Cd-name.** A cd-name identifies a Message Control System (MCS) interface area which is described in a communication description entry within the Communication Section of the Data Division. Cd-names must be unique and contain at least one alphabetic character.

   Note An MCS is application-specific and not supplied with RM/COBOL. See the *RM/COBOL User's Guide* for further information.

3. **Class-name.** A class-name identifies a user-specified list of characters. A class-name must be unique and it must contain at least one alphabetic character. A class-name is defined in the SPECIAL-NAMES paragraph of the Environment Division. It may then be used in a class condition test in the Procedure Division to determine if the current contents of a data item consist entirely of characters in the list identified by the class-name.

4. **Condition-name.** A condition-name may be defined in the SPECIAL-NAMES paragraph within the Environment Division or in a level-number 88 description within the Data Division. Condition-names must contain at least one alphabetic character.

   A SPECIAL-NAMES condition-name is assigned to ON STATUS or OFF STATUS of one of eight system software switches.

A level-number 88 condition-name 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.

A condition-name may be used in conditions as an abbreviation for the relation condition which tests whether the associated switch or conditional variable is equal to one of the set of values to which that condition-name is assigned. A condition-name may also be used in a SET statement, indicating that the associated value is to be moved to the conditional variable.

5. **Constant-name.** A constant-name is defined in a level-number 78 data description entry and names a literal value. A constant-name must be defined before any reference to the constant-name. Constant-names must contain at least one alphabetic character and must be unique. A constant-name is always global and thus may be referenced in any program contained in the program that defines the constant-name.

   An integer-valued constant-name may be defined using a constant-expression. The constant-expression is evaluated at the time of the definition during compilation and any reference to the constant-name is equivalent to a reference to the resultant integer value. The constant-expression may refer to previously defined integer-valued constant-names.

   References to constant-names may be used in any context where the assigned literal value could be used unless otherwise prohibited. The effect of a constant-name reference is the same as if the literal value assigned to the constant-name were written instead. Constant-names that have an integer value may be used wherever *integer* is specified in the syntax formats, for example, integers in BLOCK or RECORD clauses of a file control entry, integer occurrence counts in an OCCURS clause, and in constant-expressions used to define other integer-valued constant-names. An integer-valued constant-name may also be used as the integer repeat count specification in PICTURE character-strings.

6. **Data-name.** A group of contiguous characters or a numeric value treated as a unit of data is called a data item, and it is named by a data-name. A data-name must contain at least one alphabetic character. References to data items must be made unique by qualification, the appending of subscripts, or both.

   Complete unique references to data items are called identifiers. When used in the general formats, ‘data-name’ represents a word which must not be reference-modified, subscripted, or qualified unless specifically permitted by the rules of the format.

7. **File-name.** File-names are the internal names for files accessed by the source program. They are not necessarily the same as the external names by which the file is known to the runtime operating system. File-names must contain at least one alphabetic character and must be unique.

8. **Index-name.** An index-name names an index associated with a specific table. It must contain at least one alphabetic character and must be unique.

9. **Level-number.** A level-number specifies the position of a data item within a data hierarchy. A level-number is a one- or two-digit number in the range 01–49, 66, 77, 78, or 88.

   Level-numbers 66, 77, and 88 identify special properties of a data description entry.

10. **Library-name.** A library-name is a user-defined word that identifies a library to be used by the compiler for a given COPY statement. Library-names must be unique.

11. **Mnemonic-name.** A mnemonic-name is a user-defined word that is associated in the SPECIAL-NAMES paragraph with a switch-name, feature-name or low-volume-I-O-name. Mnemonic-names must be unique and must contain at least one alphabetic character.

12. **Paragraph-name.** A paragraph-name identifies the beginning of a set of COBOL procedural sentences. A reference to a nonunique paragraph-name must be made unique by qualification with a section-name.

    Paragraph-names are equivalent only if they are composed of the same sequence of the same number of digits or characters.

13. **Program-name.** The program-name identifies the source and object programs. The name must contain at least one alphabetic character.

14. **Record-name.** Record-names name data records within a file. They must contain at least one alphabetic character and, if not unique, must be made unique by qualification with the file-name.

15. **Routine-name.** A routine-name is a user-defined word that identifies a procedure written in a language other than COBOL.

16. **Screen-name.** A screen-name identifies a set of one or more entries; these entries define fields within a region of a terminal screen. Screen-names must contain at least one alphabetic character and, if not unique, must be made unique by qualification.

17. **Section-name.** A section-name identifies the beginning of a set of paragraphs. Section-names must be unique and must contain at least one alphabetic character.

18. **Segment-number.** A segment-number specifies the segmentation classification of a section. It is a one- to three-digit number in the range 00–127.

19. **Split-key-name.** A split-key-name is a user-defined word that names a concatenation of one or more data items within a record associated with an indexed file. The concatenation of the data items forms a single record key for that file. References to split-key-names must be made unique by qualification. The only qualifier allowed for a split-key-name is the file-name of the file with which the split-key-name is associated.

20. **Symbolic-character.** A symbolic-character is a user-defined word that identifies a user-defined figurative constant. Symbolic-characters must be unique and must contain at least one alphabetic character.

21. **Text-name.** A text-name is the name of a library text file. It must correspond exactly to a valid file access name that is known to the compile-time operating system.

System-Names

System-names identify certain hardware or software system components. System-names consist of code-names, device-names, feature-names, label-names, low-volume-I-O-names, record delimiting techniques, and switch-names. Most system-names are not reserved words, but certain reserved words may be used as system-names. See Table 1-2 for a complete list of system-names. See Appendix A, *Reserved Words,* for a list of system-names that are not reserved.

Table 1-2: System Names

System Name	Description
CODE-NAMES	
EBCDIC	
DEVICE-NAMES	
CARD-PUNCH	Output-only device or file
CARD-READER	Input-only device or file
CASSETTE	Input-output device or file
CONSOLE	Input-output device or file
DISC	Mass-storage device
DISK	Mass-storage device
DISPLAY	Output-only device or file
INPUT	Input-only device or file
INPUT-OUTPUT	Input-output device or file
KEYBOARD	Input-only device or file
LISTING	Print device or file
MAGNETIC-TAPE	Input-output device or file
MERGE	Sort-merge storage device
OUTPUT	Output-only device or file
PRINT	Print device or file
PRINTER	Print device or file
PRINTER-1	Print device or file
RANDOM	Mass-storage device
SORT	Sort-merge storage device
SORT-MERGE	Sort-merge storage device
SORT-WORK	Sort-merge storage device
TAPE	Input-output device or file
FEATURE-NAMES	
C01, C02, C03, . . . , C10, C11, C12	
LABEL-NAMES	
FILE-ID	Declare file access name
LABEL	Particularize label record contents
user-defined-word	Commentary

Table 1-2: System Names *(Cont.)*

System Name	Description
LOW-VOLUME-I-O-NAMES CONSOLE SYSIN SYSOUT	 Operator communication (ACCEPT, DISPLAY) Standard input (ACCEPT) Standard output (DISPLAY)
RECORD DELIMITING TECHNIQUES BINARY-SEQUENTIAL LINE-SEQUENTIAL	
SWITCH-NAMES SWITCH-1 or UPSI-0 SWITCH-2 or UPSI-1 SWITCH-3 or UPSI-2 SWITCH-4 or UPSI-3 SWITCH-5 or UPSI-4 SWITCH-6 or UPSI-5 SWITCH-7 or UPSI-6 SWITCH-8 or UPSI-7	

Reserved Words

Reserved words are those words reserved for use by the RM/COBOL compiler. A reserved word must not appear as a user-defined word within a program. Appendix A, *Reserved Words*, contains a complete list of reserved words.

Five kinds of reserved words are recognized by the compiler:

1. **Keywords.** Keywords are required elements of the formats. Their presence indicates specific compiler action.
2. **Optional Words.** Optional words are optional elements of the formats. Their presence has no effect on the object program.
3. **Connectives.** OF and IN are used interchangeably to connect qualifiers to a user-defined word. AND and OR are logical connectives, used in the formation of conditions.

4. **Special Registers.** Special registers are compiler-generated storage areas. They are used to store information that is produced in conjunction with the use of specific features. The format and description of the eight special registers are described below.

   Note The special registers may be referenced only in Procedure Division statements with the exception of the PROGRAM-ID special register.

   ADDRESS [IN / OF] *identifier-1*

   The ADDRESS special register returns the address of *identifier-1* as a pointer data item. It may only be used in certain contexts of the Procedure Division where a pointer is allowed, which are a relation condition with another pointer data item, a CALL statement USING phrase, or a Format 5 or 6 SET statement. The ADDRESS special register is not allowed in the GIVING phrase of a CALL statement even though a pointer data item is allowed there. When specified in the USING phrase of a CALL statement, the ADDRESS special register is always passed by content. When *identifier-1* is a Linkage Section data item for which the base address has not been set by being associated with an actual argument in a calling program or by execution of a SET statement, the ADDRESS special register will return a null pointer value. If *identifier-1* were referenced in such a case without the ADDRESS special register, the run unit would terminate with a data reference error. Thus, the ADDRESS special register may be used in an IF statement to prevent a data reference termination of the run unit by avoiding the reference when the ADDRESS OF *identifier-1* is equal to NULL.

   COUNT [IN / OF] *data-name-1*

   The COUNT special register exists for each COBOL table data item, that is, *data-name-1* must refer to a data item described with the OCCURS clause. For a fixed occurrence table, COUNT returns the fixed number of occurrences specified in the OCCURS clause. For a variable occurrence table, COUNT returns the value of the data-name specified by the DEPENDING ON phrase of the OCCURS clause. It may be used wherever an integer literal may be used in the Procedure Division.

   COUNT-MAX [IN / OF] *data-name-1*

   The COUNT-MAX special register exists for each COBOL table data item, that is, *data-name-1* must refer to a data item described with the OCCURS clause. COUNT-MAX always returns the maximum number of occurrences specified in the OCCURS clause. For a fixed occurrence table, COUNT, COUNT-MAX, and

COUNT-MIN will return the same value. It may be used wherever an integer literal may be used in the Procedure Division.

```
COUNT-MIN [ IN ]
          [ OF ]  data-name-1
```

The COUNT-MIN special register exists for each COBOL table data item, that is, *data-name-1* must refer to a data item described with the OCCURS clause. COUNT-MIN always returns the minimum number of occurrences specified in the OCCURS clause. For a fixed occurrence table, COUNT, COUNT-MAX, and COUNT-MIN will return the same value. It may be used wherever an integer literal may be used in the Procedure Division.

```
LENGTH [ IN ]  { identifier-1 }
       [ OF ]  { literal-1    }
```

The LENGTH special register exists for any data item or literal. It returns the length of the data item referenced by *identifier-1* or value referenced by *literal-1*. It may be used wherever an integer literal may be used in the Procedure Division. For a variable length group, the LENGTH special register returns the current length of the group. For a reference modified identifier, the LENGTH special register returns the length of the result of the reference modification, that is, the result of the evaluation of the length modifier if it was specified or the remaining length of the data item after the offset has been applied if the length modifier is not specified. For a literal, the LENGTH special register returns the number of characters in the literal. If the literal is a numeric literal, the number of characters is the same as the number of digits. That is, for a numeric literal, the sign and decimal point characters, if specified, are not counted in the length of the literal.

```
LINAGE-COUNTER [ { IN } file-name-1 ]
               [ { OF }             ]
```

The LINAGE-COUNTER special register is a line counter, generated by the presence of a LINAGE clause in a file description entry.

PROGRAM-ID

The PROGRAM-ID special register exists for any program. It returns the program-name of the program in which it is used. It may be used wherever a nonnumeric literal may be used in the program, except for the END PROGRAM header. The PROGRAM-ID special register is an exception to the rule that special registers may be referenced only in Procedure Division statements. The PROGRAM-ID special register may be specified in VALUE clauses of data description entries for nonnumeric data items or constant-name definitions. If the

program-name is specified as a nonnumeric literal in the PROGRAM-ID paragraph, the value of the PROGRAM-ID special register will match that nonnumeric literal, including its case; otherwise, the value of the PROGRAM-ID special register will be in uppercase.

RETURN-CODE

The RETURN-CODE special register has the implicit description PICTURE S9999 COMP-4, and can be set by the user to pass a return code to the calling program or the operating system before executing a STOP RUN, EXIT PROGRAM or GOBACK statement. When control is returned to a calling program, the return code passed by the called program is available to the calling program in the RETURN-CODE special register; the return code value can be tested by specifying RETURN-CODE in a relation condition. When control is returned to the operating system, the return code may be available to the command language in a system dependent manner; see the *RM/COBOL User's Guide* for specific information. The return code is initialized to zero at the start of a run unit. This is the normal return code for successful completion; other values returned are conventionally in multiples of four. Some return code values, generally the higher values, are reserved for runtime-detected errors; see the appropriate chapters on installation and system considerations in the *RM/COBOL User's Guide.*

The return code is implicitly set to the value specified in statements having the following form:

```
STOP RUN { identifier-1 }
         { integer-1    }
```

This statement is equivalent to the statement sequence:

```
MOVE { identifier-1 } TO RETURN-CODE; STOP RUN.
     { integer-1    }
```

5. **Special Characters.** The special character reserved words are the arithmetic operators (including the unary operators + and –) and relational characters:

Addition	+	Greater than/equal to	>=
Division	/	Less than	<
Equal to	=	Less than/equal to	<=
Exponentiation	**	Multiplication	*
Greater than	>	Subtraction	–

Context-Sensitive Words

The words listed in Table A-1 on page A-7 are context-sensitive words and are reserved in the specified language construct or context. If a context-sensitive word is used where the context-sensitive word is permitted in the general format, the word is treated as a keyword; otherwise, it is treated as a user-defined word.

Literals

A literal is a character-string whose representation is identical to its value. Literals are either numeric or nonnumeric.

Numeric Literals

A numeric literal represents a numeric value, not a character-string. Numeric literals are built according to the following rules:

1. The literal must contain at least 1 but not more than 30 digits.
2. The literal may contain a single + or – as the first character.
3. The literal may contain a single decimal point if the decimal point is not the last character. The decimal point must be represented with a comma if the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph.

The word *integer* when used in the syntax charts in this manual, designates an unsigned, numeric literal without a decimal point. Its value cannot be zero unless specifically allowed within a particular context.

Here are some examples:

```
1234
+1234
–1.234
.1234
+.1234
```

Nonnumeric Literals

A nonnumeric literal is a character-string enclosed in quotation marks. The character-string may contain any character from the character set of the computer. Quotation marks within the string are represented by two contiguous quotation marks. Either the quotation mark or the apostrophe may be used as the delimiter, but within one literal the first quotation mark establishes the delimiter character for that literal.

The value of the literal is the string itself excluding the delimiting character and one of each contiguous pair of embedded quotation marks. The literal may contain from 1 to 65535 characters.

Hexadecimal literals of the form:

```
H"[h]...", H'[h]...', X"[h]..."
```

or

```
X'[h]...'
```

are also permitted as another form of nonnumeric literal, where h is any valid hexadecimal digit. Two hexadecimal digits occupy one character position. If an odd number of hexadecimal digits is specified, the compiler assumes an additional hexadecimal zero digit on the right to complete the rightmost character position.

Table 1-3 lists some nonnumeric literals and their associated values.

All nonnumeric literals are of category alphanumeric.

Table 1-3: Nonnumeric Literals and Their Values

Literal	Value
"AGE"	AGE
"""TWENTY"""	"TWENTY"
'TIME'	TIME
H"4C"	4Ch
X'63B'	63B0h
""""""	Illegal (odd number of quotation marks)

Figurative Constants

Figurative constants identify commonly used constant values. These constant values are generated by the compiler according to the context in which the references occur. Note that figurative constants represent values, not literal occurrences. Thus, QUOTE cannot delimit a nonnumeric literal, SPACE is not a separator, and so forth. Singular and plural forms of figurative constants may be used interchangeably.

The following constant represents the value 0 or one or more zero characters, depending on context.

```
[ALL] ZERO, [ALL] ZEROS, [ALL] ZEROES
```

The following constant represents one or more space characters.

```
[ALL] SPACE, [ALL] SPACES
```

Except in the SPECIAL-NAMES paragraph, the following constant represents one or more occurrences of the character that has the highest ordinal position in the program collating sequence. The native HIGH-VALUE is FFh.

```
[ALL] HIGH-VALUE, [ALL] HIGH-VALUES
```

Except in the SPECIAL-NAMES paragraph, the following constant represents one or more occurrences of the character that has the lowest ordinal position in the program collating sequence. The native LOW-VALUE is 00h.

```
[ALL] LOW-VALUE, [ALL] LOW-VALUES
```

The following constant represents one or more quotation marks.

```
[ALL] QUOTE, [ALL] QUOTES
```

The following constant represents one or more null or unset pointer values. The usage of this constant is POINTER. Thus, this constant may only be used in places where a pointer literal is allowed, which are in the VALUE clause in the data description entry of a usage POINTER data item, in relation conditions involving another pointer data item, in the USING phrase of the CALL statement, and in Format 5 of the SET statement.

```
[ALL] NULL, [ALL] NULLS
```

The following constant represents all or part of the string generated by successive concatenations of the characters comprising *literal-1*. *literal-1* must be a nonnumeric literal. *literal-1* must not be a figurative constant.

ALL *literal-1*

The following constant represents one or more of the character specified as the value of *symbolic-character-1* in the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph.

[ALL] *symbolic-character-1*

When a figurative constant represents a string of one or more characters, the length of the string is determined by the compiler 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 resultant 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 that may be associated with the data item.

2. When a figurative constant, other than ALL *literal*, is not associated with another data item, as when the figurative constant appears in a DISPLAY, STOP, STRING, or UNSTRING statement, the length of the string is one character.

3. When the figurative constant ALL *literal* is not associated with another data item, the length of the string is the length of the literal.

A figurative constant may be used wherever *literal* appears in syntax, with the following exceptions:

- If the literal is restricted to a numeric literal, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

- When a figurative constant other than ALL *literal* is used, the word ALL is redundant and is used for readability only.

- If the literal is restricted to a pointer literal, the only figurative constant permitted is NULL (NULLS). NULL (NULLS) may only be used in VALUE clauses associated with a pointer data item, in relation conditions involving another pointer item, in the USING phrase of the CALL statement, in the REPLACING phrase of the INITIALIZE statement, and in Format 5 of the SET statement.

Each reserved word which refers to a figurative constant value is a distinct character-string with the exception of constructs using the word ALL, such as ALL *literal*, ALL SPACES, and so forth, which are composed of two distinct character-strings.

PICTURE Character-Strings

A PICTURE character-string consists of certain combinations of characters used as symbols. Any punctuation character appearing as part of a PICTURE character-string is considered a symbol, not a punctuation character. If the punctuation character comma, period, or semicolon is followed by a space, it is a separator which delimits the PICTURE character-string and is not part of the PICTURE character-string.

Comment-Entry

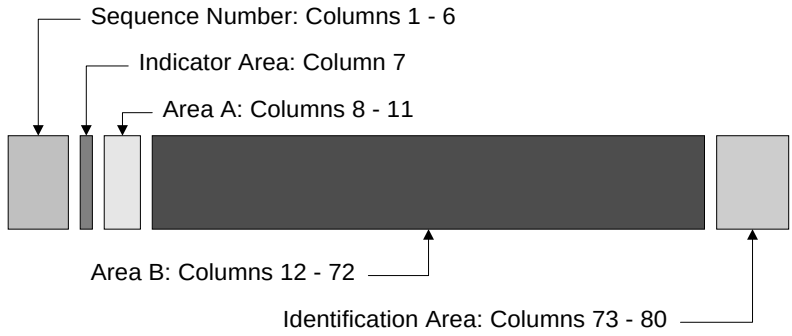
A comment-entry is an entry in the Identification Division that may contain any characters from the character set of the computer. It terminates at the next nonblank area A.

Program Structure

Source Format

Source programs are accepted as a sequence of lines (or records) of 80 characters or less. Each line is divided into five areas, as illustrated in Figure 1-1.

Figure 1-1: Source Format



The sequence number and identification areas are used for clerical and documentation purposes. They are ignored by the compiler.

The indicator area is used for denoting line continuation, comments and debugging.

Areas A and B contain the actual program according to the following rules:

1. Division headers, section headers, paragraph headers, section-names and paragraph-names must begin in area A.

2. The Data Division level indicators FD, SD, and CD, and level-numbers 01 and 77 must begin in area A. Other level-numbers may begin in area A or area B, although B is most often used.

3. The keywords, DECLARATIVES and END DECLARATIVES, precede and follow, respectively, the declaratives portion of the Procedure Division. Each must appear on a line by itself and each must begin in area A, followed by a period and a space.

4. Any other language element must begin in area B unless it immediately follows, on the same line, an element in area A.

Continuation of Lines

Any sentence, entry, phrase, or clause may be continued by starting subsequent lines in area B. These subsequent lines are called continuation lines.

The line being continued is called the continued line. Any word, literal, or PICTURE character-string may be broken in such a way that part of it appears on a continuation line, according to the following rules:

1. 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, excluding intervening comment lines or blank lines, without an 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, and the continuation line 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. The quotation mark used to continue a nonnumeric literal must be the same quotation mark (that is, it must be a quotation mark or an apostrophe) that began the nonnumeric literal.

2. If there is no hyphen in the indicator area of a line, it is assumed that the last character in the preceding line is followed by a space.

Blank Lines

A blank line is one that is blank in the indicator, A and B areas. A blank line can appear anywhere in the source program.

Comment Lines

A comment line is any line with an asterisk or a slash in the indicator area of the line. A comment line may appear as any line after the Identification Division header of a source program and as any line in library text referred to by a COPY statement. Any combination of characters from the character set of the computer may be included in area A and area B of a comment line. Comment lines are reproduced on the listing but serve as documentation only.

When a comment line is indicated with an asterisk, the comment is printed on the next available line in the listing. When a comment line is indicated with a slash, page ejection occurs before the comment line is printed.

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

In-Line Comments

An in-line comment begins with the two contiguous characters *> preceded by a separator space, and ends with the last character position of the line. An in-line comment may be placed anywhere a separator space may be placed in a COBOL source program or in library text for a COBOL source program. For the purpose of evaluating library text, pseudo-text, and source text, an in-line comment has the value of a single space character. An in-line comment that is not preceded by any COBOL words or character-strings on the same line is equivalent to a comment line, except that it may not be placed between a continued line and a continuation line if a word, literal, or PICTURE character-string is broken across the continuation.

Note An in-line comment is not recognized as such if it occurs in the sequence area (columns 1 to 6) or the identification area (columns 73 through 80) of a source line. An in-line comment that begins in the indicator area is indistinguishable from a comment line.

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 content of a debugging line must be such that a syntactically correct program is 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 Debug Compile Command Option is not specified and the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

Successive debugging lines are allowed.

A debugging line is only permitted in the separately compiled program after the OBJECT-COMPUTER paragraph, or, if the OBJECT-COMPUTER paragraph is omitted, after where the OBJECT-COMPUTER paragraph would be permitted if it were present.

Statements

Source statements always begin with a keyword called a verb. There are four kinds of statements:

1. **Directive**
2. **Conditional**
3. **Imperative**
4. **Delimited Scope**

Directive Statements

A directive statement specifies action to be taken by the compiler during compilation. The directive statements are the COPY, REPLACE, and USE statements.

Conditional Statements

A conditional 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 that specifies the AT END, NOT AT END, INVALID KEY, or NOT INVALID KEY phrase.
- A WRITE statement that specifies the INVALID KEY, NOT INVALID KEY, END-OF-PAGE, or NOT END-OF-PAGE phrase.
- A DELETE, REWRITE, or START statement that specifies the INVALID KEY or NOT INVALID KEY phrase.
- An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the ON SIZE ERROR or NOT ON SIZE ERROR phrase.
- A RECEIVE statement that specifies the NO DATA or WITH DATA phrase.
- A STRING or UNSTRING statement that specifies the ON OVERFLOW or NOT ON OVERFLOW phrase.
- A CALL statement that specifies the ON OVERFLOW, ON EXCEPTION, or NOT ON EXCEPTION phrase.

- An ACCEPT statement that specifies the ON EXCEPTION, ON ESCAPE, NOT ON EXCEPTION, or NOT ON ESCAPE phrase.

Conditional Phrases

A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from execution of a conditional statement.

A conditional phrase is one of the following:

- The AT END or NOT AT END phrase in a READ statement.
- The INVALID KEY or NOT INVALID KEY phrase in a DELETE, READ, REWRITE, START, or WRITE statement.
- The END-OF-PAGE or NOT END-OF-PAGE phrase in a WRITE statement.
- The ON SIZE ERROR or NOT ON SIZE ERROR phrase in an ADD, COMPUTE, DIVIDE, MULTIPLY, or SUBTRACT statement.
- The NO DATA or WITH DATA phrase in a RECEIVE statement.
- The ON OVERFLOW or NOT ON OVERFLOW phrase in a STRING or UNSTRING statement.
- The ON OVERFLOW, ON EXCEPTION, or NOT ON EXCEPTION phrase in a CALL statement.
- The ON EXCEPTION, ON ESCAPE, NOT ON EXCEPTION, or NOT ON ESCAPE phrase in an ACCEPT statement.

Imperative Statements

An imperative statement begins with an imperative verb and specifies an unconditional action to be taken by the object program, 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, each possibly separated from the next by a separator or the word THEN.

The imperative verbs are listed in Table 1-4.

Wherever *imperative-statement* appears in the general format of statements it refers to that sequence of consecutive imperative statements that must be ended by a period or by any phrase associated with a statement containing *imperative-statement*.

Table 1-4: Imperative Verbs

ACCEPT [1]	EXIT	REWRITE [1]
ADD [1]	GO TO	SEND
ALTER	INITIALIZE	SET
CALL [1]	INSPECT	SORT
CANCEL	MERGE	START [1]
CLOSE	MOVE	STOP
COMPUTE [1]	MULTIPLY [1]	STRING [1]
CONTINUE	OPEN	SUBTRACT [1]
DELETE [1]	PERFORM	UNSTRING [1]
DISABLE	PURGE	UNLOCK
DISPLAY	READ [1]	WRITE [1]
DIVIDE [1]	RECEIVE [1]	
ENABLE	RELEASE	

[1] *Provided no conditional phrases are present.*

Delimited Scope Statements

A delimited scope statement is any statement that includes its explicit scope terminator. The explicit scope terminators are the following:

END-ACCEPT	END-MULTIPLY	END-START
END-ADD	END-PERFORM	END-STRING
END-CALL	END-READ	END-SUBTRACT
END-DELETE	END-RECEIVE	END-UNSTRING
END-EVALUATE	END-REWRITE	END-WRITE
END-IF	END-SEARCH	

Scope of Statements

Scope terminators delimit the scope of certain Procedure Division statements. Statements that include their explicit scope terminators are called delimited scope statements. The scope of statements that are contained within statements (nested) may also be implicitly terminated.

When statements are nested within other statements, a separator period that ends the sentence implicitly terminates all nested statements.

When any statement is contained within another statement, the next phrase of the containing statement following the contained statement terminates the scope of any unterminated contained statement.

When statements are nested within other statements that allow optional conditional phrases, any optional conditional phrase encountered is considered to be the next phrase

of the nearest preceding unterminated statement with which that phrase is permitted to be associated but with which no such phrase has already been associated. An unterminated statement is one that has not been previously terminated either explicitly or implicitly.

When a delimited scope statement is nested within another delimited scope statement with the same verb, each explicit scope terminator terminates the statement started by the most recently preceding, and as yet unterminated, occurrence of that verb.

Sentences

A sentence is a sequence of one or more statements terminated by the period separator. There are three kinds of sentences:

1. A **directive sentence** may contain only a single directive statement.
2. A **conditional sentence** is a conditional statement, optionally preceded by an imperative statement, terminated by the separator period.
3. An **imperative sentence** is an imperative statement terminated by the separator period.

Clauses and Entries

An entry is an item of descriptive or declaratory nature made up of consecutive clauses. Each clause specifies an attribute of the entry. Clauses are separated by space, comma, or semicolon separators. The entry is terminated by a period separator.

Paragraphs

A paragraph is a sequence of zero, one, or more sentences or entries. In the Identification and Environment Divisions, each paragraph begins with a reserved word called a paragraph header. In the Procedure Division, each paragraph begins with a user-defined paragraph-name.

Sections

A section is a sequence of zero, one, or more paragraphs in the Environment and Procedure Divisions and a sequence of zero, one, or more entries in the Data Division. In the Environment and Data Divisions, each section begins with a section header that is made up of reserved words. In the Procedure Division, each section begins with a user-defined section-name.

Divisions

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

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

The end of a source program is indicated either by the end program header, if specified, or by the absence of additional source program lines.

Source Program General Format

The following gives the general format and order of presentation of the entries and statements that constitute a source program. The generic terms *identification-division*, *environment-division*, *data-division*, *procedure-division*, *nested-source-program*, and *end-program-header* represent an Identification Division, an Environment Division, a Data Division, a Procedure Division, a nested source program, and an end program header.

identification-division
[*environment-division*]
[*data-division*]
[*procedure-division*]
[*nested-source-program*]...
[*end-program-header*]

end-program-header must be present if either of the following circumstances exists:

- The source program contains one or more other source programs.
- The source program is contained within another source program.

General Rules

- 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.
  - An Identification Division header which indicates the start of another source program.
  - The end program header.
  - That physical position after which no more source program lines occur.
- A source program directly or indirectly contained within another program is considered in these specifications as a separate program that may additionally reference certain resources defined in the containing program.
- The object code, resulting from compiling a source program contained within another program, is considered in these specifications to be inseparable from the object code resulting from compiling the containing program.
- All separately compiled source programs in a sequence of programs must be terminated by an end program header except for the last program in the sequence.

Inter-Program Communication

The Inter-Program Communication module provides a facility by which a program can communicate with one or more programs. This communication is provided by the following:

- The ability to transfer control from one program to another within a run unit.
- The ability to pass parameters between programs to make certain data values available to a called program.

The Inter-Program Communication module also permits communication between two programs by the sharing of data and the sharing of files.

Nested Source Programs

A source program is a syntactically correct set of COBOL statements. A source program may contain other source programs; these contained programs may reference some of the resources of the program within which they are contained.

A program may be directly or indirectly contained in another program. 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 there exists a program contained in program A that also contains program B.

File Connector

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

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 with which it is associated either from within the program in which the global name is declared or from within any other program which is contained in the program which declares the global name.

A local name, however, may be used only to refer to the object with which it is associated from within the program in which the local name is declared. Some names are always global; some are always local; and some are either local or global depending upon specifications in the program in which the names are declared.

A record-name is global if the GLOBAL clause is specified in the record description entry by which the record-name is declared or, in the case of record description entries in the File Section, if the GLOBAL clause is specified in the file description entry for the file-name associated with the record description entry. A data-name is global if the GLOBAL clause is specified either in the data description entry by which the data-name is declared or in another entry to which that data description entry is subordinate. 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.

A split-key-name is global if the GLOBAL clause is specified in the file description entry for the file-name of the file with which the split-key-name is associated.

If a condition-name declared in a data description entry, a data-name, a file-name or a split-key-name is not global, the name is local.

A constant-name is always global.

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 storage associated with that object 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 object 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 that object is associated only with the program which describes the object.

External and internal objects may have 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. A file connector without the external attribute is internal to the program in which the associated file-name is described.

The data records described subordinate to a file description entry which does not contain the EXTERNAL clause or a sort-merge file description entry, as well as any data items described subordinate to the data description entries for such records, are always internal to the program 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 and Communication 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 which form part of a run unit may possess neither, one, or both of the attributes common and initial.

A common program is one which, even though it is 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 the Identification Division of the program. The COMMON clause facilitates the writing of subprograms which are 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. During the process of initializing an initial program that program's internal data is initialized; thus, 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 an undefined value. 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 Identification Division of the program.

Sharing Data in a Run Unit

Two programs in a run unit may reference common data under 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 in a Run Unit

Two programs in a run unit may reference common file connectors under the following circumstances:

- An external file connector may be referenced from any program which describes that file connector.
- 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. 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 that program describes rather than to the object possessing the same name but described in another program.

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

cd-name
paragraph-name
screen-name
section-name

The following types of user-defined words may be referenced by a program, provided that the compiler environment supports the associated library and the entities referenced are known to that system:

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 the program which contains that section:

condition-name
data-name
record-name

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

alphabet-name	mnemonic-name
class-name	symbolic-character
condition-name	

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	index-name
constant-name	program-name
data-name	record-name
file-name	split-key-name

Program-Names

A program-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 CANCEL statement and the end program header. The program-names allocated to programs constituting a run unit are not necessarily unique but, when two programs in a run unit are identically named, at least one of those two programs must be directly or indirectly contained within another separately compiled 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-name is that of a program which does not possess the common attribute and which is directly contained within another program, that program-name may be referenced only by statements included in that containing program.

2. If the program-name is that of a program which does possess the common attribute and which is directly contained within another program, that program-name may be referenced only by statements included in that containing program and any programs directly or indirectly contained within that containing program, except that program possessing the common attribute and any programs contained within it.

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

Condition-Names, Constant-Names, Data-Names, File-Names, Record-Names and Split-Key-Names

Condition-names, constant-names, data-names, file-names, record-names, and split-key-names—when declared in a source program—may be referenced only by that program except when one or more of the names are global and the program contains other programs.

See the discussion of user-defined words on page 1-4 for the requirements governing the uniqueness of the names allocated by a single program to be condition-names, constant-names, data-names, file-names, record-names, and split-key-names.

A program cannot reference any condition-name, constant-name, data-name, file-name, record-name, or split-key-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 which are 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, constant-name, a data-name, a file-name, a record-name, or a split-key-name using the same user-defined word. When such a duplicated 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 which are defined in program B and all global names which are defined in program A and in any programs which directly or indirectly contain program A. Using this set of names, the normal rules for qualification and any other rules for uniqueness of reference are applied until one or more objects are 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 has been found.

Index-Names

If a data item possessing either or both the external or global attributes includes a table accessed with an index, that index also possesses correspondingly either or both attributes. Therefore, the scope of an index-name is identical to that of the data-name which names the table whose index is named by that index-name and the scope of name rules for data-names apply. Index-names cannot be qualified.

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.

The internal data of the program contained in the Working-Storage Section and the Communication Section is initialized. If a VALUE clause is used in the description of the data item, the data item is initialized to the defined value. If a VALUE clause is not associated with a data item, the initial value of the data item is undefined.

Files with internal file connectors associated with the program are not in the open mode.

The control mechanisms for all PERFORM statements contained in the program are set to their initial states.

A GO TO statement referred to by an ALTER statement contained in the same program is set to its initial state.

A program is in the initial state:

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

End Program Header

The end program header indicates the end of the named source program.

END PROGRAM [*program-name-1* | *literal-1*].

program-name-1 must conform to the rules for forming a user-defined word (see page 1-4).

literal-1 must be a nonnumeric literal.

A constant-name may not be used for *literal-1*. A constant-name used in place of *literal-1* will be treated as a program-name; the literal value assigned to the constant-name will not be used.

program-name-1 or *literal-1* must be identical to a program-name declared in a preceding PROGRAM-ID paragraph.

If a PROGRAM-ID paragraph declaring a specific program-name is stated between the PROGRAM-ID paragraph and the end program header declaring and referencing, respectively, another program-name, the end program header referencing the former program-name must precede that referencing the latter program-name.

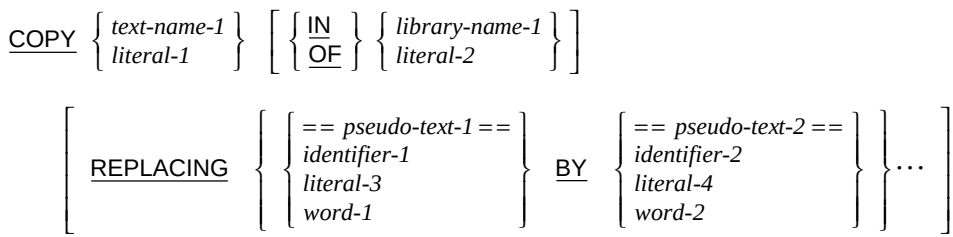
General Rules

- The end program header must be present in every program which either contains or is contained within another program.
- The end program header indicates the end of the specified source program. If *program-name-1* and *literal-1* are omitted, it is assumed to be the same as the program-name specified in the immediately preceding PROGRAM-ID paragraph not yet associated with an end program header.
- If the program terminated by the end program header is contained within another program, the next statement must either be an Identification Division header or another end program header which terminates the containing program.
- If the program terminated by the end program header is not contained within another program and if the next source statement is a COBOL statement, it must be the Identification Division header of a program to be compiled separately from the program terminated by the end program header.

COPY Statement

The COPY statement provides the facility for copying text from user-specified library files into the source program. The effect of the interpretation of the COPY statement is to insert text into the source program, where it is treated by the compiler as part of the source program.

Library text is placed in the library as a function independent of the compiler, using any text-manipulation utilities that are available. Library text must conform to the same formatting rules that apply to source text.



A constant-name may not be used for *literal-1* or *literal-2*. A constant-name used in place of *literal-1* will be treated as a text-name; the literal value assigned to the constant-name will not be used. A constant-name used in place of *literal-2* will be treated as a library-name; the literal value assigned to the constant-name will not be used.

A COPY statement may appear anywhere in a source program that a character-string or separator is allowed, except that a COPY statement may not be embedded within another COPY statement. The COPY statement may be embedded in the text referenced by the COPY statement.

A COPY statement must always be immediately followed by a period separator. That separator functions solely as a part of the COPY statement and does not terminate any sentence or entry in which the COPY statement may be embedded.

The first (or only) operand of a COPY statement may be written as a text-name or as a nonnumeric literal. If the file access name of the text file being referred to conforms to the requirements of a valid COBOL word—and it is not a reserved word—it may be written as a text-name; if it does not form a COBOL word and is made up of the following characters, it may still be a text-name:

- Alphabetic characters
- Digits (0 through 9)
- The characters ! # $ % & ( ) * – . / : ? @ \ ^ _ ‘ { }

In other words, writing the operand of a COPY statement as a nonnumeric literal is always permissible, but is required when the file access name is a reserved word, is longer than 30 characters or contains special characters other than those listed above.

In environments in which the concept of file libraries or directories has meaning, the first operand of a COPY statement may optionally be qualified by a *library-name-1*. Library-names are treated as the leading part of a file access name; the concatenation of the two values is used to locate the file to be copied. The interpretation of the concatenation of *library-name-1* and *text-name-1* is system dependent. The second operand of a COPY statement, when present, may be written as a word or as a nonnumeric literal, subject to the same considerations that apply to the first operand.

A COPY statement may be followed by additional text in area B of a source record. Multiple COPY statements may occur on a single source record.

Copy files may be nested up to five levels deep; they may contain a COPY statement. This nesting limit may be exceeded when a COPY statement appears as the last statement on the last record in a source or copy file; in such cases, the nesting level limit is raised to nine. The limit of five applies to open copy files; a COPY statement appearing at the end of a file allows the compiler to close that source or copy file before opening the one referenced in the COPY statement (that is, the compiler chains from one file to the next). The copy nesting level indicator is incremented when a COPY statement appears at the end of a file to indicate the logical nesting of the copied text. As a result, the copy level indicator does not always indicate the number of open input files and may, therefore, exceed five.

In the discussion that follows, a text word is considered a character or sequence of contiguous characters in columns 8 through 72 of records in a library, source program or in pseudo-text. These characters may be:

- A separator, except for space, a pseudo-text delimiter, and the opening and closing delimiters for nonnumeric literals. The right parenthesis and left parenthesis, regardless of context within the library, source program or pseudo-text, are always considered text words.
- A literal including, in the case of nonnumeric literals, the opening quotation mark and the closing quotation mark which bound the literal.
- Any other sequence of contiguous characters except comment lines and the word COPY, bounded by separators, which is neither a separator nor a literal.

Library text is copied into the source program without change unless a REPLACING phrase is specified. When the REPLACING phrase is specified, the following rules apply.

1. *pseudo-text-1* must contain one or more text words. It must not consist entirely of a separator comma or a separator semicolon.

2. *pseudo-text-2* may contain zero, one or more text words.

3. Character-strings within *pseudo-text-1* and *pseudo-text-2* may be continued.

4. *word-1* and *word-2* may be any single COBOL word except COPY.

5. As text is being copied from the library into the source program, each properly matched occurrence of *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3* in the library text is replaced by the corresponding *pseudo-text-2*, *identifier-2*, *word-2*, or *literal-4*.

6. For purposes of matching, *identifier-1*, *word-1*, and *literal-3* are treated as pseudo-text containing only *identifier-1*, *word-1*, or *literal-4*, respectively.

7. The comparison operation that determines text replacement is done as follows:

   a. The leftmost library text word that is not a separator comma or a separator semicolon is the first text word used for comparison. Any text word or space preceding this text word is copied into the source program. Starting with the first text word for comparison and the first *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3* that is specified in the REPLACING phrase, the entire REPLACING phrase operand that precedes the reserved word BY is compared with an equivalent number of contiguous library text words.

   b. *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3* match the library text only if the ordered sequence of text words that forms *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3* is equal, character for character, to the ordered sequence of library text words. For purposes of matching, each occurrence of a separator comma, semicolon or space in *pseudo-text-1* or in the library text is considered to be a single space. Each sequence of one or more space separators is considered to be a single space.

      For purposes of matching, a quoted string nonnumeric literal matches any other quoted string nonnumeric literal with the same value regardless of whether quotes or apostrophes were used as the delimiter. For purposes of matching, any form of a hexadecimal literal matches any other form of a hexadecimal literal that has the same value, regardless of whether an X or H is used for the initial character, whether quotes or apostrophes were used for delimiters and whether uppercase or lowercase letters are used to specify the value. A hexadecimal

literal does not match a quoted string nonnumeric literal even if the actual values would be the same in the native character set.

c. If no match occurs, the comparison is repeated with each following *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3*, if any, in the REPLACING phrase until either a match is found or there is no following REPLACING operand.

d. When all REPLACING phrase operands have been compared and no match has occurred, the leftmost library text word is copied into the source program. The following 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*, *word-1*, or *literal-3* in the REPLACING phrase.

e. Whenever a match occurs between library text and *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3*, the corresponding *pseudo-text-2*, *identifier-2*, *word-2*, or *literal-4* is placed into the source program. The library text word following the rightmost text word that participated in the match then becomes the new leftmost text word for subsequent cycles.

   The comparison cycle starts again with the first *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3* specified in the REPLACING phrase.

f. The comparison cycles continue until the rightmost text word in the library text has either participated in a match or has been considered as a leftmost library text word and participated in a complete comparison cycle.

8. Comment lines and blank lines occurring in library text or *pseudo-text-1* are ignored for purposes of matching, and the sequence of text words in the library text (if any) and in *pseudo-text-1* is determined by the rules for source format (see Figure 1-1 on page 1-18). Comment lines and blank lines appearing in *pseudo-text-2* are copied into the source program unchanged whenever *pseudo-text-2* is placed into the source program as a result of text replacement.

9. Comment lines and blank lines appearing in library text are copied into the source program unchanged except that a comment line or a blank line in library text is not copied if it appears within the sequence of text words that match *pseudo-text-1*.

10. Debugging lines may appear within library text and pseudo-text. Text words within a debugging line participate in the matching rules as if the D did not appear in 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.

11. The source program that results from the resolution of all COPY and REPLACE statements must form a syntactically correct source program, as defined in the rest of this manual.

12. Each text word copied from the library but not replaced is copied so as to start in the same area of the line in the source program as it begins in the line within the library. However, if a text word copied from the library begins in area A but follows another text word that also begins in area A of the same line, and if replacement of a preceding text word in the line by replacement text of greater length occurs, the following text word begins in area B if it cannot begin in area A. Each text word in *pseudo-text-2* that is to be placed into the source program begins in the same area of the source program as it appears in *pseudo-text-2*. Each *identifier-2*, *literal-4*, and *word-2* that is to be placed into the source program begins in the same area of the source program as the leftmost library text word that participated in the match would appear had it not been replaced.

13. If additional lines are introduced into the source program as a result of a COPY statement, each text word introduced appears on a debugging line if the COPY statement begins on a debugging line or if the text word being introduced appears on a debugging line in library text. When a text word specified in the BY phrase is introduced, it appears on a debugging line if the first library text word being replaced is specified on a debugging line. Except in the preceding cases, only those text words that are specified on debugging lines where the debugging line is within *pseudo-text-2* appear on debugging lines in the source program. If any literal specified as *literal-4* or within *pseudo-text-2* or library text is too long to be accommodated on a single line without continuation to another line in the source program, and the literal is not being placed on a debugging line, additional continuation lines are introduced to contain the remainder of the literal. A replacement literal may not be continued onto a debugging line.

14. For purposes of compilation, text words after replacement are placed in the source program according to the rules for source format (see Figure 1-1 on page 1-18). When copying text words of *pseudo-text-2* into the source program, additional spaces may be introduced between text words where there is already a space, including the space that implicitly falls between source lines.

15. If additional lines are introduced into the source program as a result of the processing of COPY statements, the indicator area of the introduced lines contains the same character as the line on which the text being replaced begins, unless that line contains a hyphen, in which case the introduced line contains a space in the indicator area. In the case where a literal is continued onto an introduced line that is not a debugging line, a hyphen is placed in the indicator area.

COPY Statement Examples

```
COPY FDFILE1.

COPY "FDFILE2.CBL".

COPY FDFILE3 OF TESTLIB.

COPY FDFILE4 IN PRODLIB.
```

REPLACE Statement

The REPLACE statement provides the ability to selectively replace source text within specified regions of the source program.

Format 1: Begin or Change Replacement)

REPLACE { == *pseudo-text-1* == BY == *pseudo-text-2* == }...

Format 2: End Replacement

REPLACE OFF

A Format 1 REPLACE statement specifies that within its scope each occurrence of *pseudo-text-1* is to be replaced by the corresponding *pseudo-text-2*.

The scope of a Format 1 REPLACE statement begins with the first text word in the source program following the REPLACE statement, and it continues up to the next REPLACE statement or the end of the program.

A Format 2 REPLACE statement terminates the scope of any preceding Format 1 REPLACE statement.

A REPLACE statement may appear anywhere in a source program that a character-string may appear. It must be preceded by a separator period except when it is the first statement in a separately compiled program. It must be terminated by a separator period.

REPLACE statements are processed after COPY statements. The text produced by the action of a REPLACE statement must not contain a REPLACE statement. The source program that results from resolution of all COPY and REPLACE statements must form a syntactically correct source program, as defined in the rest of this manual.

The word REPLACE appearing in a comment-entry or in a position where a comment-entry may appear is considered part of the comment-entry.

pseudo-text-1 must contain one or more text words. It must not consist entirely of a separator comma or a separator semicolon.

pseudo-text-2 may contain zero, one, or more text words.

Character-strings within *pseudo-text-1* and *pseudo-text-2* may be continued.

The comparison operation that determines text replacement is done as follows:

1. Starting with the leftmost text word in the scope and the first *pseudo-text-1*, *pseudo-text-1* is compared with an equivalent number of contiguous source program text words.

2. *pseudo-text-1* matches the source program text only if the ordered sequence of text words that forms *pseudo-text-1* is equal, character for character, to the ordered sequence of source program text words. For purposes of matching, each occurrence of a separator comma, semicolon 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.

   For purposes of matching, a quoted string nonnumeric literal matches any other quoted string nonnumeric literal with the same value regardless of whether quotes or apostrophes were used as the delimiter. For purposes of matching, any form of a hexadecimal literal matches any other form of a hexadecimal literal that has the same value, regardless of whether an X or H is used for the initial character, whether quotes or apostrophes were used for delimiters and whether uppercase or lowercase letters are used to specify the value. A hexadecimal literal does not match a quoted string nonnumeric literal even if the actual values would be the same in the native character set.

3. If no match occurs, the comparison is repeated with each subsequent *pseudo-text-1* until either a match is found or there is no following *pseudo-text-1*.

4. When all occurrences of *pseudo-text-1* have been compared and no match has occurred, the next source program text word in the scope is then considered as the leftmost source program text word, and the comparison cycle starts again with the first occurrence of *pseudo-text-1*.

5. 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 following the rightmost text word that participated in the match then becomes the new leftmost source program text word for subsequent cycles. The comparison cycle starts again with the first occurrence of *pseudo-text-1*.

6. The comparison cycles continue until the rightmost text word in the scope of the REPLACE statement either has participated in a match or has been considered as a leftmost source program text word and participated in a complete comparison cycle.

Comment lines and blank lines occurring in the scope or in *pseudo-text-1* are ignored for purposes of matching, and the sequence of text words in the source program text and in *pseudo-text-1* is determined by the rules for source format (see Figure 1-1 on page 1-18). Comment lines and blank lines appearing in *pseudo-text-2* are copied into the source program unchanged whenever *pseudo-text-2* is placed into the source program as a result of text replacement.

A comment or blank line in the scope is not replaced if it appears within the sequence of text words that match *pseudo-text-1*.

Debugging lines may appear within pseudo-text. Text words within a debugging line participate in the matching rules as if the D did not appear in the indicator area.

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 source format (see Figure 1-1 on page 1-18). When copying text words of *pseudo-text-2* into the source program, additional spaces may be introduced between text words where there is already a space, including the space that implicitly falls between source lines.

If additional lines are introduced into the source program as a result of the processing of REPLACE statements, the indicator area of the introduced lines contains the same character as the line on which the text being replaced begins unless that line contains a hyphen, in which case the introduced line contains a space.

If any literal within *pseudo-text-2* is too long to be accommodated on a single line without continuation to another line in the source program and the literal is not being placed on a debugging line, additional continuation lines are introduced to contain the remainder of the literal. A replacement literal may not be continued onto a debugging line.

REPLACE Statement Examples

```
REPLACE == HEADING1 == BY == FOOTING1 ==
        == HEADING2 == BY == FOOTING2 ==
        == HEADING3 == BY == FOOTING3 ==.


REPLACE == <EXEC SQL> == BY ==CALL "C$SQL" USING ==
        == <END EXEC> == BY ==GIVING SQL-STATUS. ==.


REPLACE OFF.
```

Chapter 2: Identification Division

The Identification Division must be included in every source program. This division identifies both the source program and the resulting object program. In addition, the user may include other commentary information.

This chapter details the structure and syntax of the Identification Division.

Identification Division Structure

```
{ IDENTIFICATION } DIVISION.
{ ID             }

PROGRAM-ID. { program-name-1 } [ IS { COMMON  } PROGRAM ].
            { literal-1      }       { INITIAL }

[ AUTHOR. [ comment-entry-1 ]... ]

[ INSTALLATION. [ comment-entry-2 ]... ]

[ DATE-WRITTEN. [ comment-entry-3 ]... ]

[ DATE-COMPILED. [ comment-entry-4 ]... ]

[ SECURITY. [ comment-entry-5 ]... ]

[ REMARKS. [ comment-entry-6 ]... ]
```

comment-entry may be any combination of characters from the character set of the computer. The continuation of *comment-entry* by the use of the hyphen in the indicator area is not permitted; however, *comment-entry* may be contained on one or more lines. A *comment-entry* must be contained in area B of a source line and is ended by source text in area A of a source line. A COPY or REPLACE statement within a *comment-entry* is considered part of the *comment-entry* and has no effect on the resultant source program.

Program Identification

```
{ IDENTIFICATION } DIVISION.
{ ID             }
```

The Identification Division must begin with the reserved words IDENTIFICATION DIVISION or ID DIVISION followed by a separator period.

Paragraph headers identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be written in any order.

PROGRAM-ID Paragraph

```
PROGRAM-ID. { program-name-1 } [ IS { COMMON } PROGRAM ].
            { literal-1      }       { INITIAL }
```

A constant-name may not be used for *literal-1*. A constant-name used in place of *literal-1* will be treated as a program-name; the literal value assigned to the constant-name will not be used.

The PROGRAM-ID paragraph, containing the program-name, identifies the source program, the object program, and all listings pertaining to a particular program. *program-name-1* is a user-defined word. Alternatively, *program-name-1* may be specified as a nonnumeric literal, in which case the value of *program-name-1* may be a reserved word or may use any characters in the character set of the computer. A program contained within another program must not be assigned the same name as that of any other program contained within the separately compiled program that contains this program.

program-name-1 may be 1 to 30 characters in length. All the characters of *program-name-1*, except trailing spaces, are associated with the object program in order to identify the program to be called or canceled by a CALL or CANCEL statement.

The PROGRAM-ID paragraph also assigns selected program attributes to the program that it names.

The optional COMMON clause may be used only if the program is contained within another program. It specifies that the program is common. A common program is contained within another program, but may be called from programs other than that containing it. Such other calling programs must be directly or indirectly contained in the same program that contains the common program.

The INITIAL clause specifies that the program is initial. When an initial program is called, it and any programs contained within it are placed in their initial state. When an EXIT PROGRAM or GOBACK statement is executed in an initial program, the program is implicitly canceled.

AUTHOR, INSTALLATION, DATE-WRITTEN, SECURITY, and REMARKS Paragraphs

AUTHOR. [*comment-entry-1*]...

INSTALLATION. [*comment-entry-2*]...

DATE-WRITTEN. [*comment-entry-3*]...

SECURITY. [*comment-entry-5*]...

REMARKS. [*comment-entry-6*]...

These paragraphs are optional; their order of presentation is immaterial. They document information pertaining to the paragraph header. The paragraphs are reproduced in the listing generated by the compiler, but have no effect on the compilation.

DATE-COMPILED Paragraph

DATE-COMPILED. [*comment-entry-4*]...

If a DATE-COMPILED paragraph is present, it is replaced during compilation with a paragraph of the form:

```
DATE-COMPILED.  current-date.
```

where *current-date* is the date on which the compilation started. The format of *current-date* is determined by the LISTING-DATE-FORMAT and LISTING-DATE-SEPARATOR keywords of the COMPILER-OPTIONS configuration record. The default format is “MM/DD/YYYY”, where MM is the month of the year, DD is the day of the month, and YYYY is the year.

The entire *comment-entry-4* is replaced, but comment lines in the paragraph are not replaced. Only the compilation listing file is affected; the compilation date is not inserted in the source file. The inserted compilation date matches the date placed in the object file and the date listed in the compilation listing page headers.

The DATE-COMPILED paragraph is optional and may appear in any order with respect to the other optional paragraphs of the Identification Division.

Chapter 3: Environment Division

The Environment Division describes the hardware configuration of the compiling (or source) computer and the computer on which the object program is run (the object computer). It also describes the relationship between the files and the input-output media.

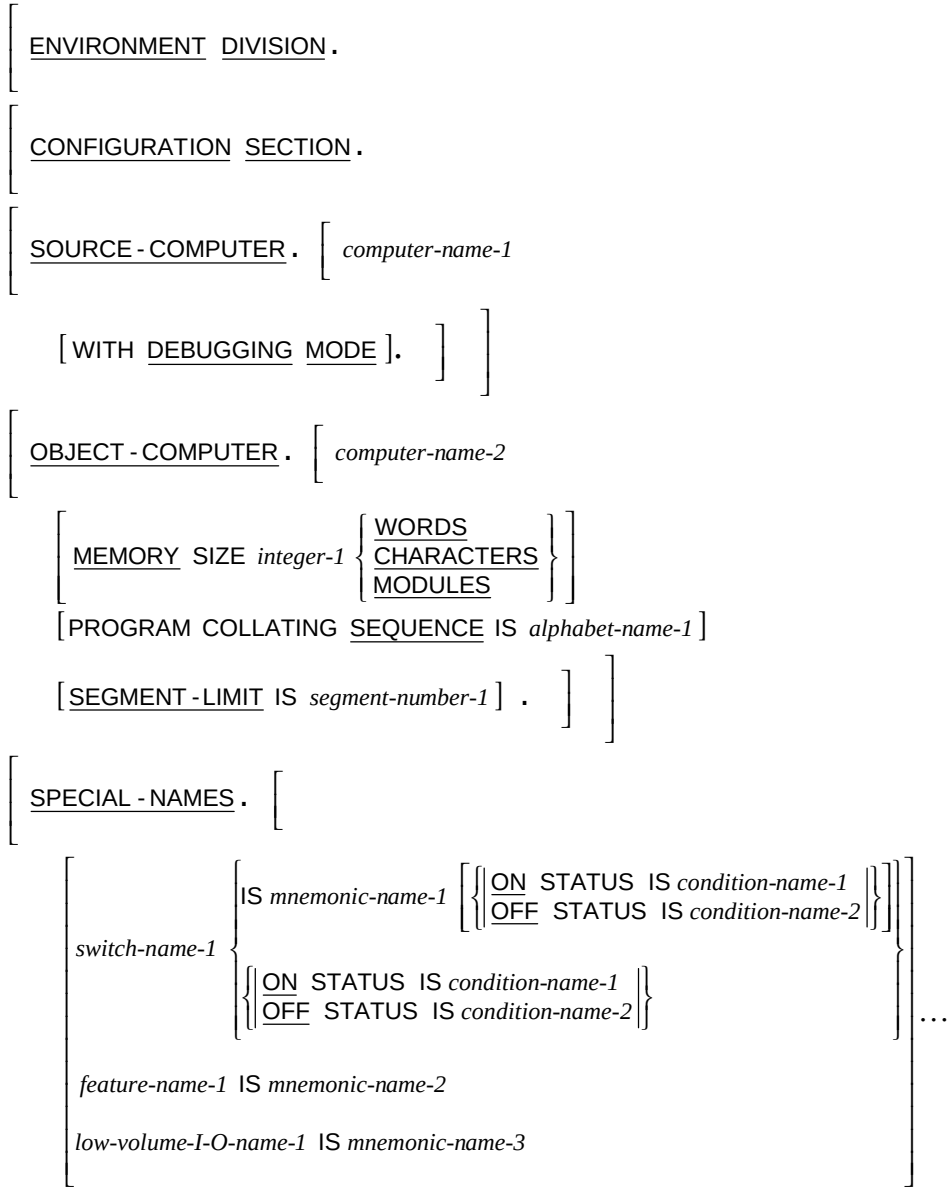
The Environment Division is an optional division in a source program. It is subdivided in two sections.

Environment Division Structure

The two sections in the Environment Division are as follows:

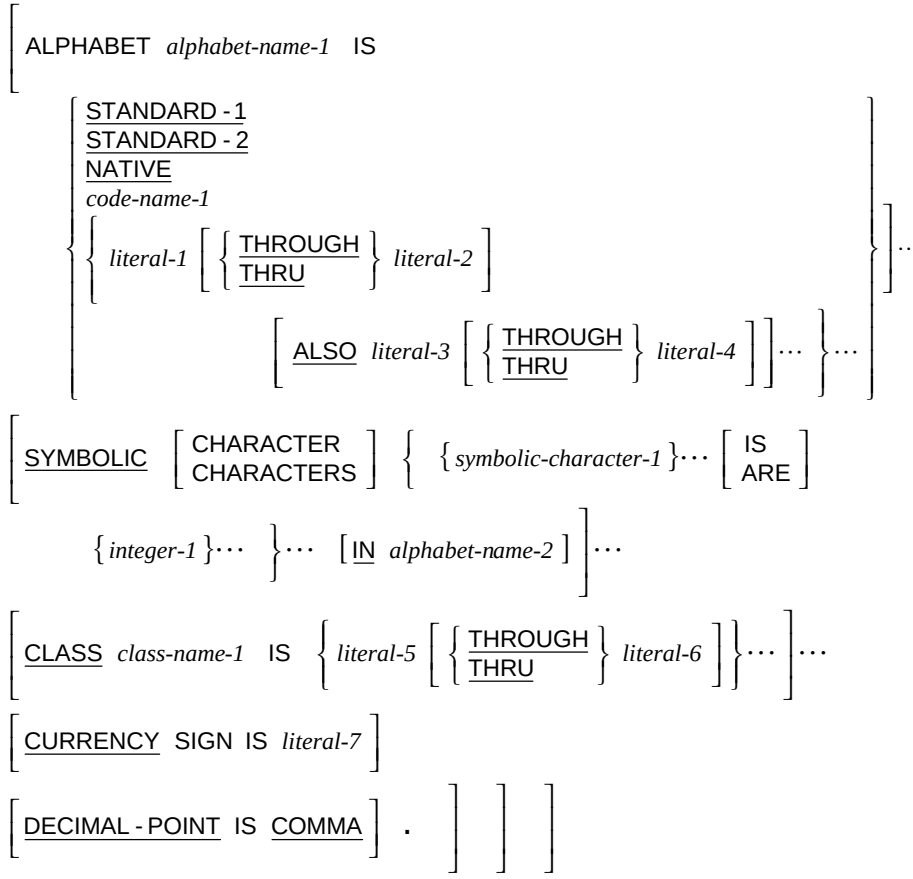
1. The **Configuration Section**, which describes the overall specifications of source and object programs (see page 3-5).

2. The **Input-Output Section**, which names the files and external media required by an object program and which provides information required for transmission and handling of data during running of the object program (see page 3-18).



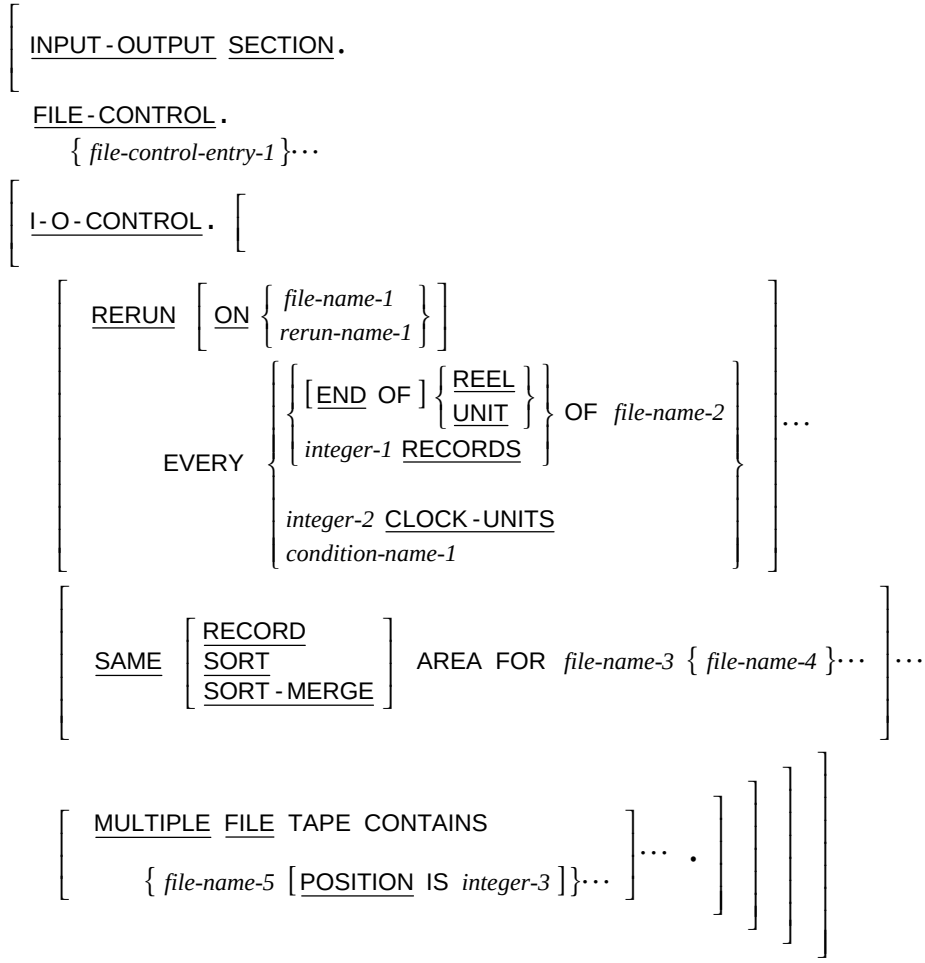
(continued on next page)

(continued from previous page)



(continued on next page)

(continued from previous page)



Configuration Section

The Configuration Section deals with the characteristics of the source computer and the object computer. This section is divided into three paragraphs:

1. The **SOURCE-COMPUTER paragraph**, which describes the computer configuration on which the source program is compiled (see the next section).

2. The **OBJECT-COMPUTER paragraph**, which describes the computer configuration on which the object program produced by the compiler is to be run (see page 3-6).

3. The **SPECIAL-NAMES paragraph**, which relates names used by the compiler to user-defined words in the source program (see page 3-7).

The Configuration Section must not be stated in a program which is contained within another program.

The entries explicitly or implicitly stated in the Configuration Section of a program that contains other programs apply to each contained program.

SOURCE-COMPUTER Paragraph

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

SOURCE-COMPUTER. [*computer-name-1* [WITH DEBUGGING MODE].]

computer-name-1 is a user-defined word and is only commentary.

All clauses of the SOURCE-COMPUTER paragraph apply to the program in which they are explicitly or implicitly specified and to any program contained within that program.

If the WITH DEBUGGING MODE clause is used, all debugging lines (D in the indicator area, column 7) are compiled as if there were a blank in the indicator area.

If the WITH DEBUGGING MODE clause is not specified, any debugging lines (D in the indicator area, column 7) are compiled as if they were comment lines unless the Debug compilation option is specified (see the *RM/COBOL User's Guide* for details).

When the Debug compilation option is specified, debugging lines are compiled as if there were a blank in the indicator area whether or not the WITH DEBUGGING MODE phrase is specified in the source programs.

When multiple programs are compiled with one invocation of the compiler without the Debug option, the WITH DEBUGGING MODE phrase may be used in one or more of the source programs without affecting other source programs compiled in the same group.

OBJECT-COMPUTER Paragraph

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

```
OBJECT-COMPUTER. [ computer-name-2
    [ MEMORY SIZE integer-1 { WORDS      } ]
                            { CHARACTERS }
                            { MODULES    }
    [PROGRAM COLLATING SEQUENCE IS alphabet-name-1 ]
    [SEGMENT-LIMIT IS segment-number-1 ] . ]
```

computer-name-2 is a user-defined word and is only commentary.

All clauses of the OBJECT-COMPUTER paragraph apply to the program in which they are explicitly or implicitly specified and to any program contained within that program.

The MEMORY SIZE clause is treated as commentary.

The PROGRAM COLLATING SEQUENCE clause specifies the program collating sequence to be used in determining the truth value of any nonnumeric comparisons. If the PROGRAM COLLATING SEQUENCE clause is specified, the program collating sequence is the collating sequence associated with *alphabet-name-1*. If the PROGRAM COLLATING SEQUENCE clause is not specified, the collating sequence is ASCII.

The program collating sequence established in the OBJECT-COMPUTER paragraph determines the truth value of any nonnumeric comparisons that are as follows:

- Explicitly specified in relation conditions.
- Explicitly specified in condition-name conditions.

The program collating sequence established in the OBJECT-COMPUTER paragraph is applied to any nonnumeric merge or sort keys unless the COLLATING SEQUENCE phrase is specified in the respective SORT or MERGE statement.

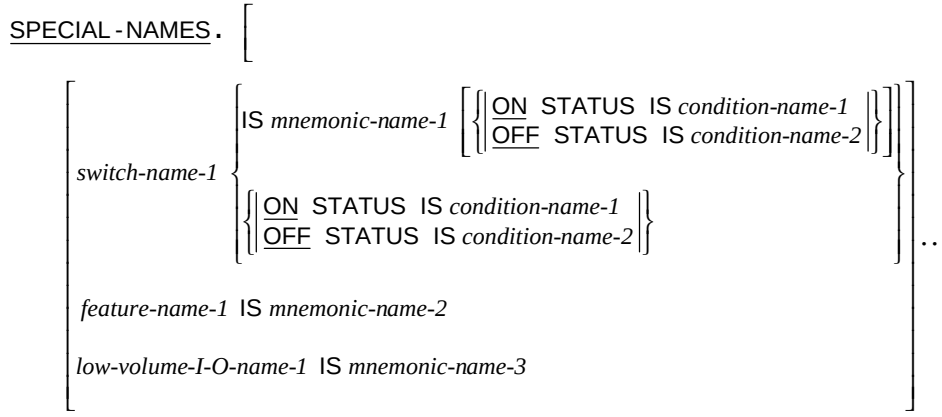
The SEGMENT-LIMIT clause allows the user to reduce the number of permanent segments in the program, while still retaining the logical properties of fixed portion segments (segment-numbers 0 through 49). When the SEGMENT-LIMIT clause is specified, only those segments having segment-numbers from 0 up to, but not including, the segment-number designated as the segment-limit, are considered as permanent segments of the object program. *segment-number-1* must be an integer from 1 to 49.

If the SEGMENT-LIMIT clause is omitted, all segments having segment-numbers 0 through 49 are considered permanent segments of the object program.

The clauses of the OBJECT-COMPUTER paragraph may appear in any order.

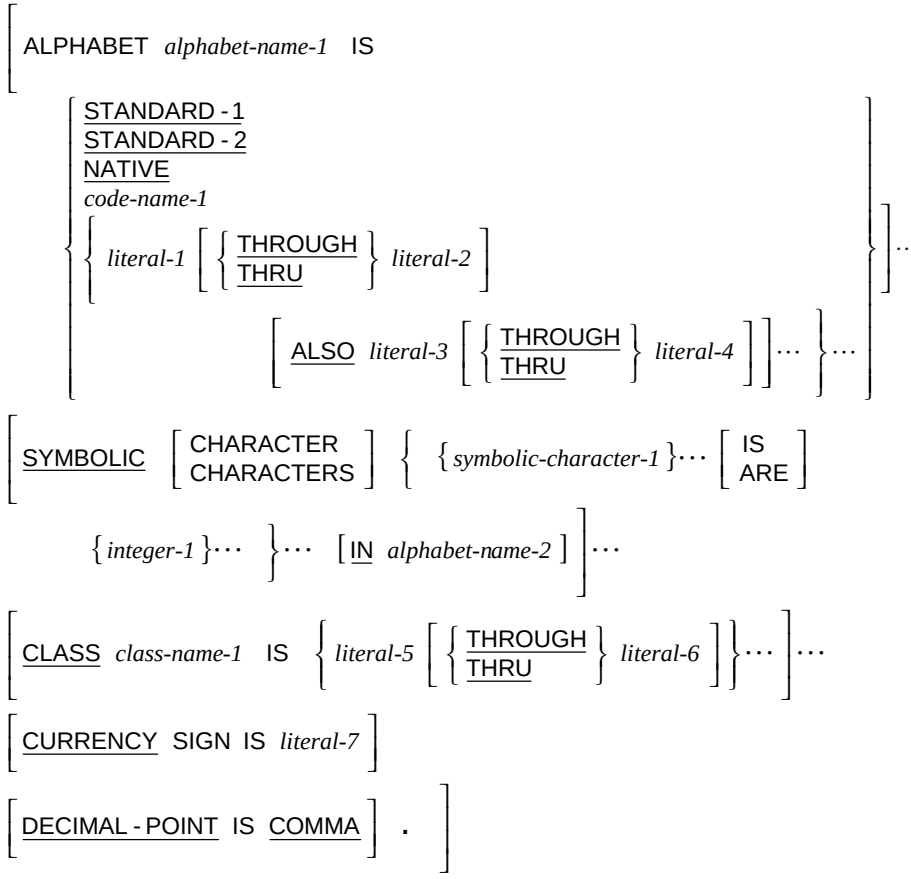
SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph relates names used by the compiler to user-defined words in the source program.



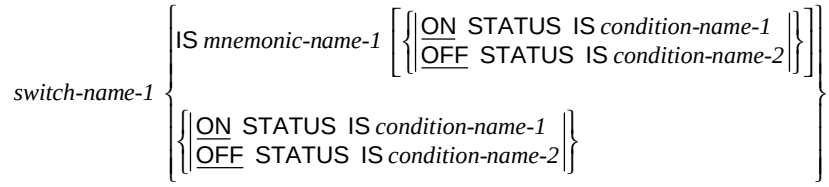
(continued on next page)

(continued from previous page)



All clauses specified in the SPECIAL-NAMES paragraph for a program also apply to programs contained within that program. The alphabet-names, class-names, condition-names, and symbolic-characters specified in the SPECIAL-NAMES paragraph of the containing program may be referenced from any contained program.

Mnemonic-Name Clause



feature-name-1 IS *mnemonic-name-2*

low-volume-I-O-name-1 IS *mnemonic-name-3*

The mnemonic-name clause provides a means to relate names to switches, features, and low-volume I-O devices available in the implementation.

switch-name may be SWITCH-1, SWITCH-2, . . ., SWITCH-8 or UPSI-0, UPSI-1, . . ., UPSI-7. Switch-names UPSI-0 through UPSI-7 are synonymous with switch-names SWITCH-1 through SWITCH-8.

The status of any switch may be altered by the execution of a Format 3 SET statement which specifies as its operand the mnemonic-name associated with that switch.

Zero, one or two condition-names may be defined with each switch-name entry. Condition-names defined in this way become associated with the ON or OFF status of a switch and may be used in condition-name tests in the Procedure Division to interrogate the current setting of the switch.

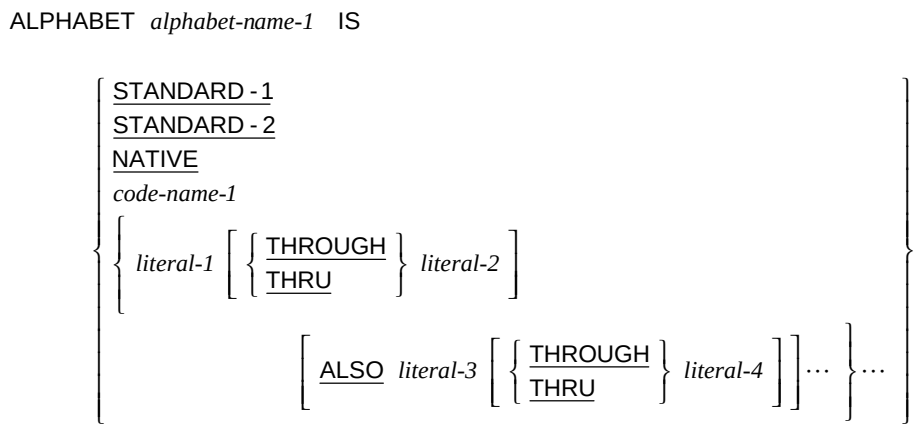
feature-name-1 may be any of the channel-names C01, C02, . . ., C12. The feature-name entries may be used to associate mnemonic-names with specific channel-names. The mnemonic-names may then be used in WRITE and SEND statements to control vertical positioning on a hard-copy printing device. The actual effect of the various channel-names is hardware-dependent and is, therefore, defined in the *RM/COBOL User's Guide*.

low-volume-I-O-name-1 may be CONSOLE, SYSIN, or SYSOUT. CONSOLE is the primary terminal (keyboard and screen) associated with the run unit of which this program is a part. SYSIN is the standard input file for the run unit which may be the keyboard of the primary terminal. SYSOUT is the standard output file for the run unit which may be the screen of the primary terminal.

mnemonic-name-1, *mnemonic-name-2* and *mnemonic-name-3* are user-defined words. Their meaning is defined in the SPECIAL-NAMES paragraph, as shown above. Once defined, they may be used in certain contexts within the Procedure Division, as follows:

- *mnemonic-name-1* becomes the name of a particular switch; it may be used only in SET statements.
- *mnemonic-name-2* becomes a reference to a feature-name. It may be used only in SEND and WRITE statements.
- *mnemonic-name-3* becomes a reference to the associated low-volume-I-O-name. It may be used only in ACCEPT and DISPLAY statements.

ALPHABET Clause



The ALPHABET clause provides a means for relating a name to a specified character code set or collating sequence. When the alphabet-name is referenced in the PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph, the COLLATING SEQUENCE clause of an Indexed File Control Entry or the COLLATING SEQUENCE phrase of a SORT or MERGE statement, the ALPHABET clause specifies a collating sequence. When the alphabet-name is referenced in a SYMBOLIC CHARACTERS or CODE-SET clause, the ALPHABET clause specifies a character code set.

If the STANDARD-1 phrase is specified, the character code set or collating sequence identified is that defined in *American National Standard X3.4-1977, Code for Information Interchange*, usually referred to as ASCII. If the STANDARD-2 phrase is specified, the character code set identified is the International Reference Version of the ISO 7-bit code defined in International Standard 646, 7-bit Coded Character Set for Information Processing Interchange. If the NATIVE phrase is specified, the native

character set or collating sequence is used. If the code-name phrase is specified with the code-name EBCDIC, the character code or collating sequence is the extended binary coded decimal interchange code (8 bits, no parity). See Appendix J, *Code-Set Translation Tables*, in the *RM/COBOL User's Guide*.

If the literal form of the ALPHABET clause is specified, the following rules apply:

1. A given character must not be specified more than once in an ALPHABET clause that is referenced in the PROGRAM COLLATING SEQUENCE clause, the COLLATING SEQUENCE clause in the File-Control entry, the COLLATING SEQUENCE phrase of the SORT and MERGE statements, or that is associated with a code set for a file that is opened in either the extend, I-O or output mode.

2. A given character may be specified more than once in an ALPHABET clause only if *alphabet-name-1* is referenced in a SYMBOLIC CHARACTERS clause or is associated with a code set for a file that is opened in the input mode.

literal-1, *literal-2*, *literal-3*, *literal-4*, *literal-5* and *literal-6* must not specify a symbolic-character figurative constant. When a literal in an ALPHABET clause or CLASS clause is numeric, it must be an unsigned integer and its value must be in the range 1 to 256, inclusive. When a literal in an ALPHABET clause or CLASS clause is nonnumeric and it is in a THROUGH or ALSO phrase, it must be one character in length.

The character that has the highest ordinal position in the current program collating sequence is associated with the figurative constant HIGH-VALUE, except when this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. If more than one character has the highest position in the program collating sequence, the last character specified is associated with the figurative constant HIGH-VALUE.

The character that has the lowest ordinal position in the current program collating sequence is associated with the figurative constant LOW-VALUE, except when this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. If more than one character has the lowest position in the program collating sequence, the first character specified is associated with the figurative constant LOW-VALUE.

When specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with those characters having the highest and lowest positions, respectively, in the native collating sequence.

The collating sequence identified is that defined according to the following rules:

1. If the literal is numeric, it specifies the ordinal number of a character within the native character set. If the literal is single-character nonnumeric, it specifies the actual character within the native character set. If the literal is multiple-character nonnumeric, 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 that are not explicitly specified in the literal phrase assume a position (in the collating sequence being specified) that is greater than any of the explicitly specified characters. The relative order within the set of these unspecified characters is the same as the native collating sequence order.

4. If the THROUGH (or THRU) 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 values of *literal-1* and *literal-3* are assigned to the same position in the collating sequence being specified.

The ALPHABET clause of the SPECIAL-NAMES paragraph defines three different character set mappings:

1. An output code set mapping of native characters to external characters.

2. An input code set mapping of external characters to native characters.

3. A collating sequence mapping of characters to character positions.

Which of these mappings is intended depends on the use made of the defined alphabet. The input or output code set mapping is indicated by the CODE-SET clause of the SELECT or FD entry; the input code set mapping is indicated by the SYMBOLIC CHARACTERS . . . IN *alphabet-name* clause of the SPECIAL-NAMES paragraph. The collating sequence mapping is indicated by the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph, the COLLATING SEQUENCE clause of the SORT and MERGE statements, and by the COLLATING SEQUENCE clause of the SELECT entry of an indexed organization file.

Code Name Alphabets

RM/COBOL supports four code-names in the ALPHABET clause: NATIVE, STANDARD-1, STANDARD-2 and EBCDIC.

The NATIVE alphabet always represents the 256 character code values possible in the computer. The graphic equivalents of these character code values may be ASCII or EBCDIC, depending on the source computer. The chosen native code set is recorded in the object program.

The STANDARD-1 alphabet contains 128 characters in the range 00h to 7Fh. This alphabet is defined in the document *American National Standard X3.4-1977, Code for Information Interchange* and is commonly referred to as ASCII. If the native character set is ASCII, the 128 ASCII characters are represented by the identical values 00h to 7Fh in the native character set, and native characters 80h to FFh have no STANDARD-1 equivalent. If the native character set is EBCDIC, the 128 ASCII characters are represented by the corresponding 128 native EBCDIC values, and the remaining 128 EBCDIC values have no STANDARD-1 equivalent.

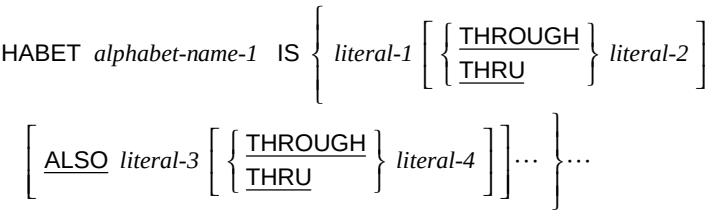
The STANDARD-2 alphabet is the same as the STANDARD-1 alphabet except for the currency symbol character.

The EBCDIC alphabet contains 256 characters, 128 of which have widely accepted standard ASCII equivalents. For the purpose of processing the SYMBOLIC CHARACTERS clause when the native code set is based on ASCII, all 256 EBCDIC character codes are assigned ASCII equivalents. See the *RM/COBOL User's Guide* for the exact mappings used to effect these conversions.

Literal Alphabets

RM/COBOL supports user-defined literal alphabets for file code sets and for program, sort-merge, and indexed file collating sequences. One use for a literal code set would be to map all lowercase letters to uppercase on input or output to a file. Another would be to specify a different ASCII to EBCDIC mapping than that built into RM/COBOL. A literal collating sequence could be used to cause lowercase letters in indexed file keys to be treated as uppercase, or to cause numbers to follow letters in indexed file keys. Europeans might use a literal collating sequence to cause the correct ordering of keys which contain letters not in the English alphabet.

The syntax for defining a literal alphabet is:



The value of *literal-1* is the ordinal position or value of a native character. The ordinal position of *literal-1* in the list of literals is the collating position when the alphabet is used as a collating sequence, and is one greater than the binary value of the external character code when used as a code set. The ALSO phrase allows more than one native character to have the same collating position or be translated to the same external character.

For example, the following alphabet causes lowercase and uppercase native characters to be collated to the same position:

```
ALPHABET OUT-UPPER IS 1 THRU 65,
  "A" ALSO "a", "B" ALSO "b", "C" ALSO "c", "D" ALSO "d",
  "E" ALSO "e", "F" ALSO "f", "G" ALSO "g", "H" ALSO "h",
  "I" ALSO "i", "J" ALSO "j", "K" ALSO "k", "L" ALSO "l",
  "M" ALSO "m", "N" ALSO "n", "O" ALSO "o", "P" ALSO "p",
  "Q" ALSO "q", "R" ALSO "r", "S" ALSO "s", "T" ALSO "t",
  "U" ALSO "u", "V" ALSO "v", "W" ALSO "w", "X" ALSO "x",
  "Y" ALSO "y", "Z" ALSO "z", 92 THRU 97, 124 THRU 128;
```

The alphabet OUT-UPPER, when used as a code set of a file opened for output, causes lowercase characters in the records being written to be replaced by uppercase characters.

The final phrase—124 THRU 128—is redundant when the alphabet is used as a collating sequence, since unspecified characters are collated in their natural order following the last specified character. If any characters are omitted from the definition of the alphabet and the characters occur in a record being written, a file status 97 will result. The following alphabet causes lowercase external characters to be converted to uppercase native characters on file input:

```
ALPHABET IN-UPPER IS   1 THROUGH  65,
   "A" THROUGH "Z",   92 THROUGH  97,
   "A" THROUGH "Z",  124 THROUGH 128;
```

An alphabet in which a native character occurs more than once may be used only on a file opened for input or in the SYMBOLIC CHARACTERS clause. Such an alphabet is an illegal collating sequence and is an illegal code set on a file opened for output, extend or I-O.

Indexed File Alphabets

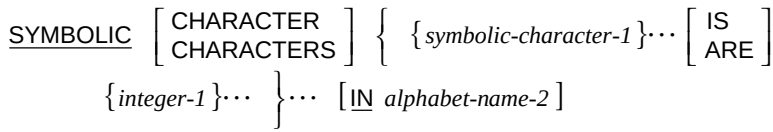
RM/COBOL accepts both the CODE-SET and COLLATING SEQUENCE clauses when defining an indexed organization file. The CODE-SET clause can be used on an ASCII object computer to read an IBM EBCDIC ISAM file; the runtime system then performs EBCDIC to ASCII translation of data read and ASCII to EBCDIC translation of data written. The COLLATING SEQUENCE clause can be used to force lowercase and uppercase key values to be treated identically, or to cause a more natural ordering of European characters with diacritical marks.

When the CODE-SET clause is specified and the COLLATING SEQUENCE clause is omitted, the natural collating sequence of the external character set is used. To put it another way: if the COLLATING SEQUENCE is omitted, the alphabet referred to in the CODE-SET clause is used, and the native collating sequence is used if the CODE-SET clause is also omitted.

EBCDIC Translation

The *RM/COBOL User's Guide* contains an appendix (named "Code-Set Translation Tables") that defines the translation between the ASCII and EBCDIC character sets. The ASCII to EBCDIC translation is identical to that described by IBM in the document *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic* (SC30-3112-0, March 1976). The EBCDIC to ASCII translation is the inverse of the ASCII to EBCDIC mapping, with the addition that EBCDIC characters with no ASCII equivalent are assigned values in the range 80h to FFh.

SYMBOLIC CHARACTERS Clause



The SYMBOLIC CHARACTERS clause provides the ability to define named figurative constants above and beyond those that are standard in the language. Such additional figurative constants are named by the symbolic-character, which is a user-defined word. A given symbolic-character may not be defined more than once in a program. In the SYMBOLIC CHARACTERS clause, the relationship between each symbolic-character and the corresponding integer is by position; that is, 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. There must be a one-to-one correspondence between occurrences of *symbolic-character-1* and occurrences of *integer-1*.

If there is no IN *alphabet-name-2* clause immediately following *integer-1*, *integer-1* specifies the ordinal position of *symbolic-character-1* in the native character set; otherwise, *integer-1* specifies the ordinal position of *symbolic-character-1* in the character set identified by *alphabet-name-2*.

The ordinal position specified by *integer-1* must exist in the native character set. If the IN phrase is specified, the ordinal position must exist in the character set named by *alphabet-name-2*.

The internal representation of *symbolic-character-1* is the internal representation of the character represented in the native character set.

The SYMBOLIC CHARACTERS clause without the IN alphabet phrase associates an identifier with a native character. *integer-1* of the format is the position of the ASCII or EBCDIC code rather than the code itself. Position has an offset of 1 from the value of the code. The *RM/COBOL User's Guide* contains an appendix (named "Code-Set Translation Tables") that shows the ASCII and EBCDIC character positions.

For example:

```
SYMBOLIC CHARACTERS NAK-CHARACTER IS 22;
```

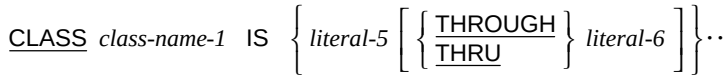
This clause achieves its intended result only if the native character set is ASCII. If the native character set is EBCDIC, NAK-CHARACTER still receives the value of position 22, but the value is interpreted as a newline character.

The following clauses define an EBCDIC NAK character:

```
ALPHABET EBCDIC-ALPHABET IS EBCDIC;
SYMBOLIC CHARACTERS NAK-CHARACTER IS 62 IN EBCDIC-ALPHABET;
```

If the native character set is EBCDIC, the identifier EBCDIC-NAK is associated with the value 62 (3Dh plus 1). If the native character set is ASCII, the identifier EBCDIC-NAK is associated with the value 22.

CLASS Clause



The CLASS clause provides a means of assigning a name to the specified set of characters listed in that clause. *class-name* can be referenced only in a class condition in the Procedure Division. The characters specified by the values of the literals in this clause define the exclusive set of characters of which this class-name consists. The CLASS clause defines class conditions other than those that are standard to the language.

For each numeric literal in the list, the value of the literal specifies the ordinal number of a character within the native character set. This value must not exceed the value which represents the number of characters in the native character set.

For each nonnumeric literal in the list, the value of the character or characters in the literal specifies the actual character or characters within the native character set. When a nonnumeric literal is used in a THROUGH phrase, it must be a single-character literal.

If the THROUGH phrase is specified, the contiguous characters in the native character set beginning with the character specified by the value of *literal-5*, and ending with the character specified by the value of *literal-6*, are included in the set of characters identified by *class-name*. In addition, the contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending order.

CURRENCY SIGN Clause

CURRENCY SIGN IS *literal-7*

The literal that appears in the CURRENCY SIGN clause is used in the PICTURE clause to represent the currency symbol. The literal must be nonnumeric and is limited to a single character. The value of the literal must *not* be any of the following characters:

- Alphabetic characters A, B, C, D, P, R, S, V, X, Z or the space
- Digits 0 through 9
- Special characters: * + – , . ; ( ) ” / =

If the CURRENCY SIGN clause is specified, then both the default currency sign ($) and the currency symbol (cs) specified in the CURRENCY SIGN clause may be used in PICTURE character-strings in that source program, although they are mutually exclusive in any one PICTURE character-string. The values of the currency sign and currency symbol may be changed at execution time by runtime configuration as explained in the *RM/COBOL User's Guide*. If CURRENCY SIGN IS “$” is specified, then “$” is the currency symbol and there is no currency sign.

If the CURRENCY SIGN clause is not specified, only the currency sign ($) is used in PICTURE character-strings and there is no currency symbol.

DECIMAL-POINT Clause

DECIMAL - POINT IS COMMA

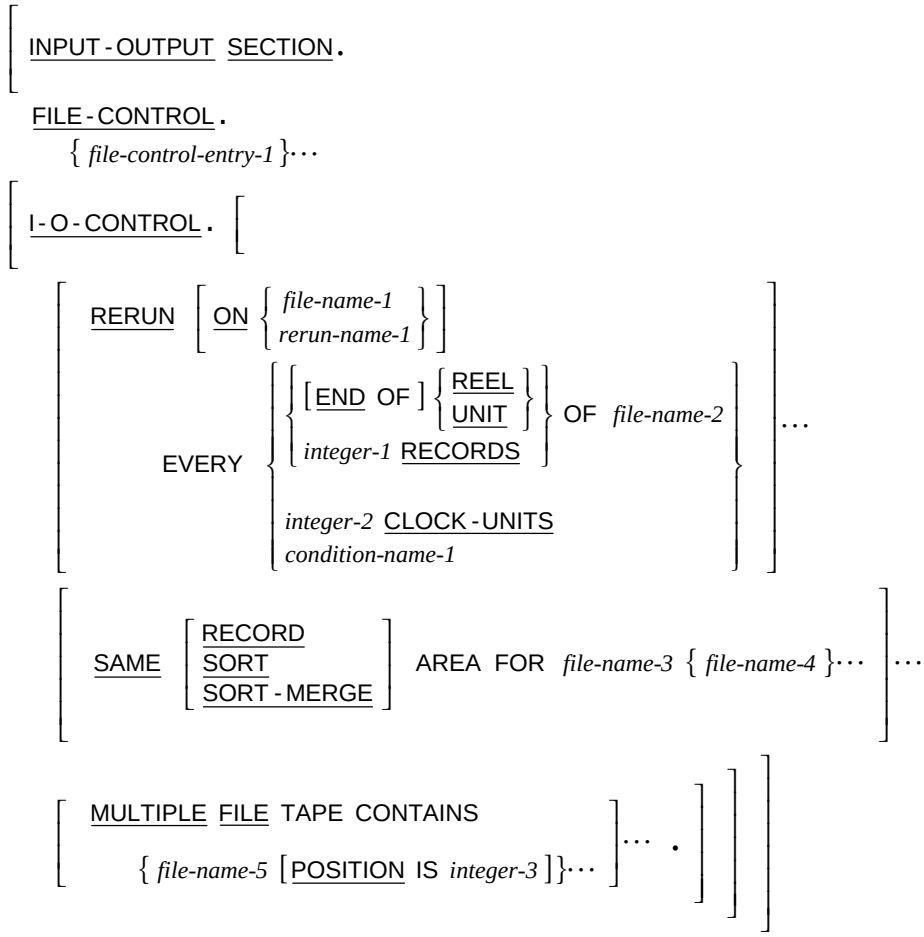
The DECIMAL-POINT IS COMMA clause declares that the function of the comma and period are exchanged in the character-string of the PICTURE clause, in numeric literals, and in conversion of numeric data for the ACCEPT and DISPLAY statements. The value of the decimal point and comma characters may be changed at execution time by runtime configuration regardless of the presence of this clause as explained in the *RM/COBOL User's Guide*.

Input-Output Section

The Input-Output section names the files and external media required by an object program and provides information required for transmission and handling of data during execution of the object program. This section is divided into two paragraphs:

1. The **FILE-CONTROL paragraph**, which names and associates the files with external media (see page 3-19).

2. The **I-O-CONTROL paragraph**, which defines special control techniques to be used in the object program (see page 3-36).



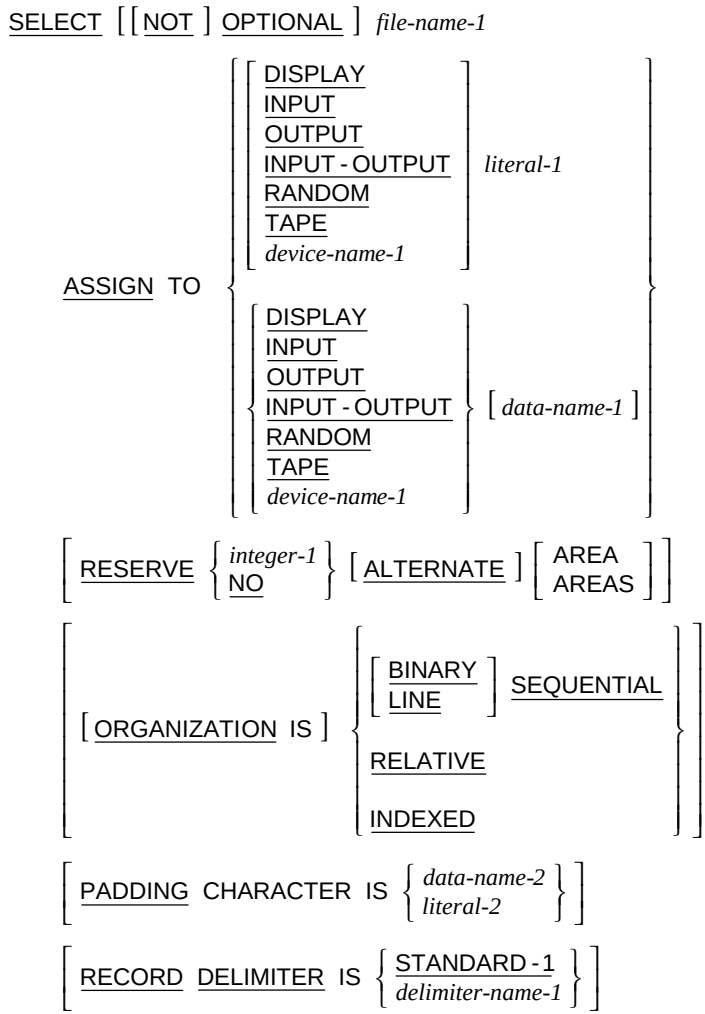
FILE-CONTROL Paragraph

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

The content of *file-control-entry-1* depends on the organization of the file named. In addition, there is a separate form for a sort-merge file.

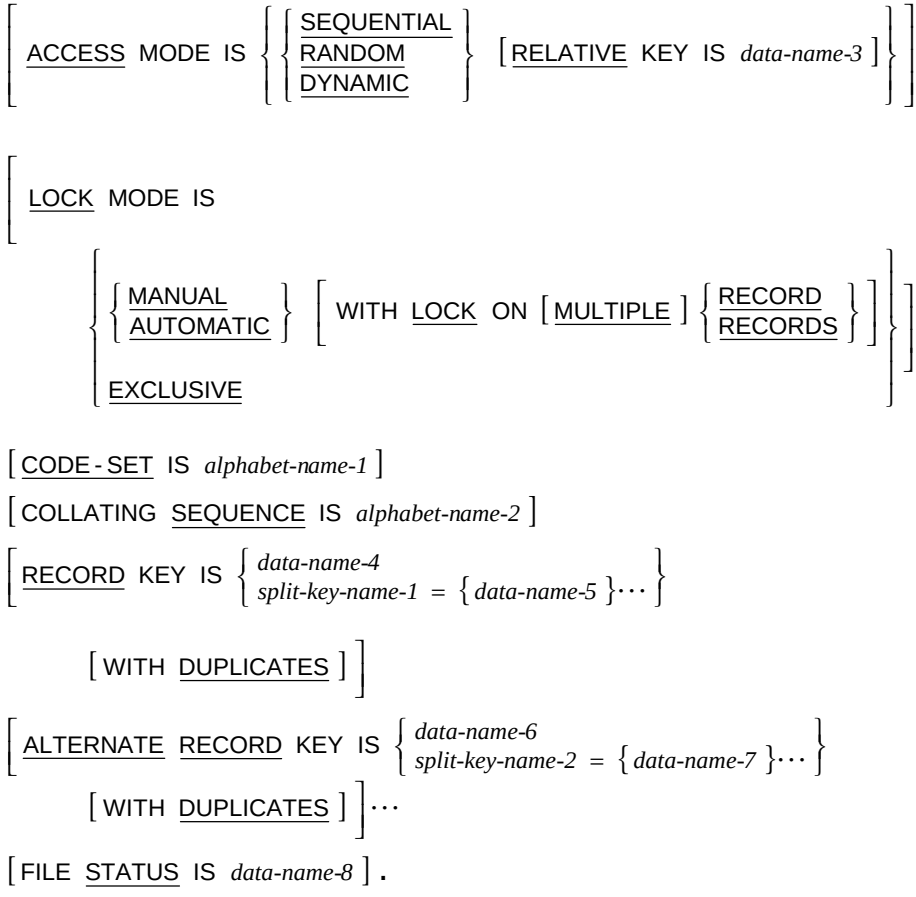
File Control Entry

The file control entry names a sequential, relative, or indexed organization file and provides other file-related information.



(continued on next page)

(continued from previous page)



SELECT Clause

SELECT [[NOT] OPTIONAL] *file-name-1*

The SELECT clause must be specified first in the file control entry. The clauses that follow may appear in any order. (These other clauses are discussed in alphabetical order on the following pages.)

If the file connector referenced by *file-name-1* is an external file connector, all file control entries in the run unit which reference this file connector must have:

- The same specification for the OPTIONAL phrase.
- A consistent specification for *device-name-1* in the ASSIGN clause. The file access name specified in the ASSIGN clause, *literal-1* or *data-name-1*, or in the VALUE OF clause should also be consistent, but the file access name specified by the program that executes the OPEN statement for *file-name-1* will be used.
- The same RECORD DELIMITER specification.
- The same value for *integer-1* and the same presence or absence of the ALTERNATE phrase in the RESERVE clause.
- The same organization.
- The same access mode.
- The same lock mode.
- The same character set for the CODE-SET clause.
- The same specification for the PADDING CHARACTER clause.
- The same external data item for *data-name-3* in the RELATIVE KEY phrase.
- The same collating sequence for the COLLATING SEQUENCE clause.
- The same data description entry for *data-name-4* or each *data-name-5*, the same number of *data-name-5* in the definition of *split-key-name-1*, the same relative location within the associated record for *data-name-4* or each *data-name-5*, and the same presence or absence of the DUPLICATES phrase.
- The same data description entry for *data-name-6* or each *data-name-7*, the same number of *data-name-7* in the definition of *split-key-name-2*, the same relative location within the associated record for *data-name-6* or each *data-name-7*, the same presence or absence of the DUPLICATES phrase, and the same number of alternate record keys.

The OPTIONAL phrase applies to files opened in input, I-O, or extend modes. Its specification is required for files that may not be present each time they are opened for input, I-O, or extend.

The NOT OPTIONAL phrase is redundant commentary because, by default, files are not optional, that is, files are required to be present each time they are opened for input, I-O, or extend. The phrase is supported only for compatibility with other COBOL dialects that include this phrase.

Each file described in the Data Division must be named once and only once as *file-name-1* in the FILE-CONTROL paragraph. Each file specified in a file control entry must have a file description entry in the Data Division of the same program.

ACCESS MODE Clause

The ACCESS MODE clause specifies the order in which records are to be accessed in the file. If the ACCESS MODE clause is not specified, ACCESS MODE IS SEQUENTIAL is implied.

If the access mode is sequential, records in the file are accessed in the sequence dictated by the file organization:

- For sequential files, this sequence is specified by predecessor-successor record relationships established by the execution of WRITE statements when the file is created or extended.
- For relative files, this sequence is the order of ascending or descending relative record numbers of existing records in the file.
- For indexed files, this sequence is ascending or descending within a given key of reference according to the collating sequence of the file.

If the access mode is random, records in the file are accessed according to a key dictated by the file organization:

- For sequential files, random access may not be specified.
- For relative files, this key is the value of the relative key data item specified by *data-name-3* in the RELATIVE KEY phrase. The RELATIVE KEY phrase is required when RANDOM is specified in the ACCESS MODE clause for a relative file.
- For indexed files, this key is the value of a record key data item for the file. The random access mode is not recommended for indexed files that are described with the DUPLICATES phrase in the RECORD KEY clause. If the DUPLICATES phrase is specified in the RECORD KEY clause of the file control entry, then DELETE and

REWRITE statements are not allowed in the random access mode, and READ statements can only access the first of a set of records with the same prime record key value.

If the access mode is dynamic, records in the file may be accessed sequentially or randomly as described in the rules for the input-output statements. Dynamic access may not be specified for sequential organization files. The RELATIVE KEY phrase is required when DYNAMIC is specified in the ACCESS MODE clause for a relative file.

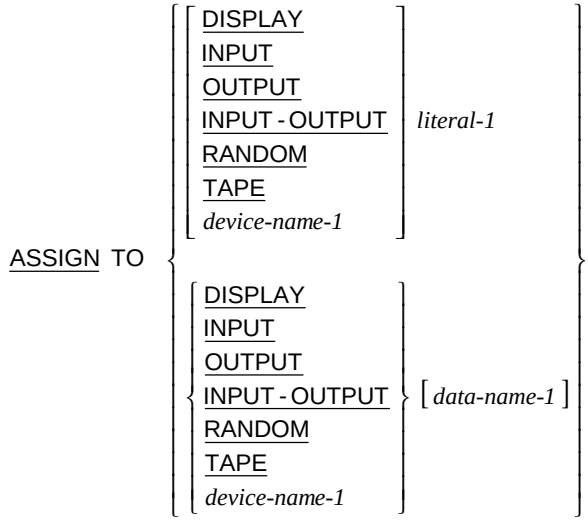
The RELATIVE KEY phrase may only be specified in the ACCESS MODE clause of a file control entry that describes a relative organization file. If the access mode is random or dynamic, the RELATIVE KEY phrase must be specified within the ACCESS MODE clause for a relative file. Also, if a relative file is referenced in a START statement, the RELATIVE KEY phrase within the ACCESS MODE clause must be specified for that file. The relative key data item associated with the execution of an input-output statement for a relative file is the data item referenced by *data-name-3* in the RELATIVE KEY phrase of the ACCESS MODE clause.

All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4, and so forth.

The data item specified by *data-name-3* is used to communicate a relative record number between the user and the mass storage control system (MSCS). *data-name-3* may be qualified. *data-name-3* must reference an unsigned integer data item whose description does not contain the PICTURE symbol 'P'. *data-name-3* must not be defined in a record description entry associated with *file-name-1*.

If the associated file connector is an external file connector, every file control entry in the run unit that is associated with that file connector must specify the same access mode. In addition, for relative files, *data-name-3* must reference an external data item and the RELATIVE KEY phrase in each associated file control entry must reference that same external data item in each case.

ASSIGN Clause



The ASSIGN clause specifies the association of the file referenced by *file-name-1* with a class of external storage devices as indicated by *device-name-1*. For sequential organization files, there are 18 permissible selections for *device-name-1*. They are CARD-PUNCH, CARD-READER, CASSETTE, CONSOLE, DISC, DISK, DISPLAY, INPUT, INPUT-OUTPUT, KEYBOARD, LISTING, MAGNETIC-TAPE, OUTPUT, PRINT, PRINTER, PRINTER-1, RANDOM, and TAPE. For relative and indexed organization files, one of the mass storage device names (DISC, DISK, or RANDOM) must be specified.

The contexts in which *file-name-1* is used in the rest of the program establish these constraints on the *device-name-1* that may be assigned:

1. If the file is used in an OPEN I-O statement, or if a record of the file is used in a REWRITE statement, *device-name-1* must be DISC, DISK or RANDOM. In this context these words are synonymous.

2. If the file is used in an OPEN INPUT or READ statement, or if it appears in the USING list of a SORT or MERGE statement, *device-name-1* must be CARD-READER, CASSETTE, CONSOLE, DISC, DISK, INPUT, INPUT-OUTPUT, KEYBOARD, MAGNETIC-TAPE, RANDOM or TAPE.

3. If the file is used in an OPEN EXTEND or OPEN OUTPUT statement, if it appears in the GIVING list of a SORT or MERGE statement, if it is used in a RERUN ON phrase, or if a record of the file is used in a WRITE statement, *device-name-1* must be CARD-PUNCH, CASSETTE, CONSOLE, DISC, DISK, DISPLAY,

INPUT-OUTPUT, MAGNETIC-TAPE, OUTPUT, PRINT, PRINTER, PRINTER-1, RANDOM or TAPE.

The ASSIGN clause may also specify the file access name with *literal-1* or as the contents of a data item identified by *data-name-1*. The file access name is the name used to identify the physical file when the program is run. See the file description entry VALUE OF clause on page 4-17 for an alternative method of specifying the file access name. If neither the ASSIGN clause nor the VALUE OF clause specify a file access name, then *file-name-1* is used for the file access name. In any case, the value of the file access name must be valid according to operating system dependent rules for identifying a file or device. If the file access name is specified by a literal in the program, portability is more likely if the file access name is short (eight or fewer characters) and contains only letters and digits. Most operating systems provide a means to map such file access names to the longer names necessary to identify a particular physical file. See the *RM/COBOL User's Guide* for information on mapping file access names at execution time.

If *literal-1* is specified, it must be a nonnumeric literal. In this case, *device-name-1* may be omitted and the compiler will infer the storage device type from the organization of the file and the I-O statements used in the program. If *file-name-1* refers to an external file connector for a sequential file, the compiler will assume a mass storage device when *device-name-1* is omitted.

If *data-name-1* is specified, it must be defined in the Data Division as a data item of the category alphanumeric. The value of this data item is used as the file access name at the time an OPEN statement is executed for the file. If *data-name-1* refers to a variable length group, the maximum size of the group will be used to determine the file access name, independent of the value of the DEPENDING ON data item. *data-name-1* may be qualified.

CODE-SET Clause

CODE-SET IS *alphabet-name-1*

The CODE-SET clause specifies the character code convention used to represent data on the external medium. That external character code convention may or may not be the same as the internal native character code convention.

When there is a CODE-SET clause associated with a file, and its *alphabet-name-1* specifies a code-set other than the native code-set, then for each record of the file that is read from or written to the external medium a character-by-character translation is done to convert the text of the record according to the mapping specified by *alphabet-name-1*.

If there is no CODE-SET clause associated with a file, or if there is a CODE-SET clause and its *alphabet-name-1* specifies the native code-set, the external character code

convention for the file is the same as the internal code convention, and no character translation is done.

A CODE-SET clause for a file may be specified either in the file control entry for the file (as shown in the format), or in the file description entry for the file in the Data Division. It is permissible to specify a CODE-SET clause in both places, but both alphabet-names must be the same.

If the associated file connector is an external file connector, all CODE-SET clauses in the run unit that are associated with that file connector must have the same character set.

In some runtime environments the identity of the code-set associated with a file at the time it is created is preserved with the file as one of its fixed attributes. In such environments it may be a requirement that each time the file is subsequently opened the code-set associated with the file be the same as its original code-set. See the *RM/COBOL User's Guide* for more specific information.

COLLATING SEQUENCE Clause

COLLATING SEQUENCE IS *alphabet-name-2*

The COLLATING SEQUENCE clause may be used to specify a character mapping to be used on the values of the keys of an indexed file before determining their ordering. If no COLLATING SEQUENCE clause is present, the keys are ordered according to the collating sequence implied by the explicitly specified or default code-set of the file. When this clause is used, *alphabet-name-2* must have been defined as an alphabet-name in the SPECIAL-NAMES paragraph of the Configuration Section in the Environment Division. The character code set designated by *alphabet-name-2* determines the ordering of the keys of the file. Support for a specified collating sequence is system-dependent. On those systems that do not support a specific collating sequence, the native collating sequence is used. See the *RM/COBOL User's Guide* for details.

The COLLATING SEQUENCE clause may only be specified in a file control entry that describes an indexed organization file.

In some runtime environments the specification of a COLLATING SEQUENCE clause has no effect. In other runtime environments the identity of the collating sequence associated with a file at the time it is created is preserved with the file as one of its fixed attributes. In such environments it may be a requirement that each time the file is subsequently opened, the collating sequence specified for the file be the same as its original collating sequence. See the *RM/COBOL User's Guide* for more specific information on this point.

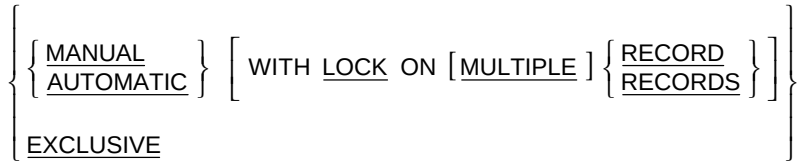
FILE STATUS Clause

FILE STATUS IS *data-name-8*

When the FILE STATUS clause is specified, a value will be moved by the runtime system into the data item specified by *data-name-8* after the execution of every statement that references the file either explicitly or implicitly. This value indicates the status of execution of the statement. *data-name-8* must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section. *data-name-8* may be qualified. The data item referenced by *data-name-8* that is updated during the execution of an input-output statement is the one specified in the file control entry associated with that statement.

LOCK MODE Clause

LOCK MODE IS



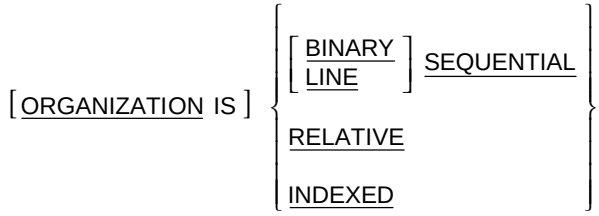
The LOCK MODE clause specifies whether a file is to be opened in exclusive or shared lock mode and, if shared, the record locking mode. If the LOCK MODE clause is omitted in the file control entry, the file sharing lock mode for the file is determined by options in the OPEN statement, the environment in which the file is opened and a configurable default (see the topic, "File Sharing," and the FORCE-USER-MODE configuration keyword in the *RM/COBOL User's Guide).* The default record locking mode for shared files opened for input-output (open I-O mode) is automatic single.

- The EXCLUSIVE phrase indicates that all OPEN statements that refer to *file-name-1* are to open the file in exclusive mode.
- The MANUAL phrase indicates that an OPEN statement without the EXCLUSIVE or an applicable WITH LOCK phrase for *file-name-1* is to open the file in shared mode and, if the open mode is I-O, in one of the manual record locking modes.
- The AUTOMATIC phrase indicates that an OPEN statement without the EXCLUSIVE or an applicable WITH LOCK phrase for *file-name-1* is to open the file in shared mode and, if the open mode is I-O, in one of the automatic record locking modes.

- The LOCK ON RECORD phrase specifies one of the single record locking modes. Single record locking modes apply when AUTOMATIC or MANUAL is explicitly stated without the MULTIPLE option.
- The LOCK ON MULTIPLE RECORDS phrase specifies one of the multiple record locking modes.

See "File Locking" on page 5-68 for a description of file locking modes. See "Record Locking" on page 5-69 for a description of record locking modes. If the associated file connector is an external file connector, every file control entry in the run unit that is associated with that file connector must specify the same lock mode.

ORGANIZATION Clause



The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed. When the ORGANIZATION clause is not specified, ORGANIZATION IS SEQUENTIAL is implied.

Sequential

Sequential organization is a permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

Sequential files may be further classified by the record delimiting technique used to determine the length of records in the file. The ORGANIZATION clause may specify the record delimiting technique to be binary sequential with the BINARY option or line sequential with the LINE option. For additional information on record delimiting techniques, see the description of the RECORD DELIMITER clause on page 3-31.

Relative

Relative organization is a 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.

Indexed

Indexed organization is a permanent logical file structure in which each record is identified by the value of one or more keys within that record. All records are uniquely identified by the value of the prime record key, except when the DUPLICATES phrase is specified in the RECORD KEY clause. Alternate record keys may be defined to provide alternate access paths to records in an indexed file. Record keys may be split keys, which are a concatenation of a sequence of data items that are not necessarily contiguous within the record.

PADDING CHARACTER Clause

PADDING CHARACTER IS { *data-name-2* / *literal-2* }

The PADDING CHARACTER clause provides a way to specify the character that is used to fill out or pad blocks for sequential files. If the padding character is defined with a data-name, *data-name-2* may be qualified. It must refer to a one-character data item of the category alphanumeric defined in the Working-Storage or Linkage Section. If the padding character is defined with a literal, *literal-2* must be a one-character nonnumeric literal.

The PADDING CHARACTER clause may only be specified in a file control entry that describes a sequential organization file.

literal-2 or the value of the data item referenced by *data-name-2* at the time the file is opened for output is used as the value of the padding character, and this value becomes a fixed attribute of the file.

During input operations on a file whose file control entry includes a PADDING CHARACTER clause, any portion of a block that exists beyond the last logical record and consists entirely of padding characters is bypassed, and a logical record that consists entirely of padding characters is ignored. During output operations on such a file, any portion of a block that exists beyond the last logical record is filled out with padding characters.

The use and recognition of padding characters occur only if such operations are compatible with the supporting device type. See the *RM/COBOL User's Guide* for more information on this point.

If the associated file connector is an external file connector, all PADDING CHARACTER clauses in the run unit which are associated with that file connector must have the same specifications. If *data-name-2* is specified, it must reference an external data item.

RECORD DELIMITER Clause

RECORD DELIMITER IS { STANDARD-1 / *delimiter-name-1* }

The RECORD DELIMITER clause specifies the record delimiting technique for a sequential file. The record delimiting technique determines how records are separated on the external medium. An alternative method of specifying the record delimiting technique is the LINE or BINARY option of the ORGANIZATION clause. The record delimiting technique is established at the time a file is created and cannot subsequently be changed.

The RECORD DELIMITER clause may only be specified in a file control entry that describes a sequential organization file.

The RECORD DELIMITER clause with the BINARY-SEQUENTIAL option specifies that the file record delimiting technique is binary sequential. The binary sequential record delimiting technique uses record length headers and trailers to delimit each variable-length record on the external medium. This allows binary sequential files to contain data items with usage other than DISPLAY. For fixed-length binary sequential records, no record delimiter is needed or used. All characters in the records of a binary sequential file are treated as data, not as control characters. When the BINARY-SEQUENTIAL option is specified, the ORGANIZATION clause must not specify the LINE option.

The RECORD DELIMITER clause with the LINE-SEQUENTIAL option specifies that the file record delimiting technique is line sequential. The line sequential record delimiting technique is defined to be the same as that used by the standard system text editor. Typically, this record delimiting technique uses special control characters to delimit each record, for example, a carriage-return line-feed pair. Therefore, such files should contain only data items that are explicitly or implicitly defined with USAGE IS DISPLAY. If there are data items with usage other than DISPLAY in a line sequential file, their values may be interpreted as control characters, for example, record separators or horizontal tabs. When the LINE-SEQUENTIAL option is specified, the ORGANIZATION clause must not specify the BINARY option.

The use of the RECORD DELIMITER clause with the STANDARD-1 option is meaningful only when the supporting external medium is magnetic tape. When this is the case, the clause may be used to indicate that the method of determining the length of a variable record on the external medium is as specified in American National Standard X3.27-1978, Magnetic Tape Labels and File Structure for Information Interchange and International Standard 1001 1979, Magnetic Tape Labels and File Structure for Information Interchange. See the *RM/COBOL User's Guide* for more information on this point. The RECORD DELIMITER clause with the STANDARD-1 option may not be specified if LINE or BINARY is specified in the ORGANIZATION clause since they each specify a different record delimiting technique.

If the RECORD DELIMITER clause is not specified and neither the LINE nor BINARY option is specified in the ORGANIZATION clause, the record delimiting technique for the file is determined by the presence of a Compile Command option or a Runtime Command option. See the *RM/COBOL User's Guide* for further information on these options.

If the associated file connector is an external file connector, all RECORD DELIMITER clauses in the run unit which are associated with that file connector must have the same specifications.

RECORD KEY and ALTERNATE RECORD KEY Clauses

RECORD KEY IS { *data-name-4* / *split-key-name-1* = { *data-name-5* }... } [WITH DUPLICATES]

ALTERNATE RECORD KEY IS { *data-name-6* / *split-key-name-2* = { *data-name-7* }... }

[WITH DUPLICATES]

The RECORD KEY clause specifies the record key that is the prime record key for an indexed file. The values of the prime record key must be unique among records of the file, except when the DUPLICATES phrase is specified in the RECORD KEY clause. This prime record key provides an access path to records in an indexed file. *split-key-name-1* names a concatenation of one or more data items within a record associated with the file. The concatenation of the data items, which need not be contiguous within the record, forms a single record key. *split-key-name-1* may be specified only in a READ or START statement.

An ALTERNATE RECORD KEY clause specifies a record key that is an alternate record key for an indexed file. This alternate record key provides an alternate access path to records in an indexed file. Up to 254 alternate record keys may be declared for an indexed organization file. *split-key-name-2* names a concatenation of one or more data items within a record associated with the file. The concatenation of the data items, which need not be contiguous within the record, forms a single record key. *split-key-name-2* may be specified only in a READ or START statement.

Note There is a limit of 255 key segments per indexed file. Thus, if split keys are used, the limit of 254 alternate keys is reduced accordingly.

The RECORD KEY and ALTERNATE RECORD KEY clauses may only be specified in a file control entry that describes an indexed organization file.

The RECORD KEY clause is required in a file control entry that describes an indexed organization file.

If the associated file connector is an external file connector, every file control entry in the run unit that is associated with that file connector must specify the same data description entry for *data-name-4*, *data-name-6* or each *data-name-5*, *data-name-7*, the same number of *data-name-4*, *data-name-6* in the definition of *split-key-name-1*, *split-key-name-2*, the same relative location within the associated record for *data-name-4*, *data-name-6* or each *data-name-5*, *data-name-7*, the same presence or absence of the DUPLICATES phrase, and the same number of alternate record keys.

The data descriptions of *data-name-4*, *data-name-5*, *data-name-6*, and *data-name-7*, as well as their relative locations within a record, must be the same as those used when the file was created. The number of alternate keys for the file, the sequence of *data-name-5* or *data-name-7* for each key, and the presence or absence of the DUPLICATES phrase for each key must also be the same as when the file was created.

The data items to which *data-name-4*, *data-name-5*, *data-name-6*, and *data-name-7* refer must each be defined within a record description entry associated with *file-name-1*. Each data item must also be defined either as a category alphanumeric data item or as an unsigned integer data item with DISPLAY usage.

None of *data-name-4*, *data-name-5*, *data-name-6*, and *data-name-7* may be described as a data item whose size is variable.

data-name-5 cannot refer to an item whose leftmost character position corresponds to the leftmost character position of an item to which *data-name-4* or any other *data-name-5* associated with this file refers. *split-key-name-2* cannot specify a list of data-names that results in the same key as *split-key-name-1* or any other *split-key-name-2*. Two record keys are considered the same if they have the same relative offset within the record for each key segment, the same length for each key segment and the same number of key segments, where a key segment corresponds to a single data item in the concatenation of data items that form the split key. If *split-key-name-2* lists only one *data-name-7*, then *data-name-7* cannot refer to an item whose leftmost character position and length correspond to the leftmost character position and length of an item to which *data-name-4* or any *data-name-5* associated with this file refers.

data-name-4, *data-name-5*, *data-name-6*, and *data-name-7* may be qualified.

The DUPLICATES phrase specifies that the value of the associated 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 record key must not be duplicated among any of the records in the file. When the DUPLICATES phrase is specified in the RECORD KEY clause, the value of the prime record key is not necessarily a unique identifier for a single record; therefore, in this case, the DELETE and REWRITE statements are disallowed in the random access mode and are sequential operations in the dynamic access mode.

Note The ALTERNATE RECORD KEY clauses may be specified in any order within the file control entry. The compiler sorts the alternate keys into ascending order of offset within the associated record and then ascending length of key segment. For two or more keys with the same offset and length of key segment, the keys are sorted into ascending number of segments. The compiler produces an error if two or more keys are the same, that is, they have the same relative location of each segment, the same length for each segment, and the same number of segments. This sorting of the alternate keys ensures that the associated indexed file description is independent of the order in which ALTERNATE RECORD KEY clauses are specified in the programs that refer to an indexed file.

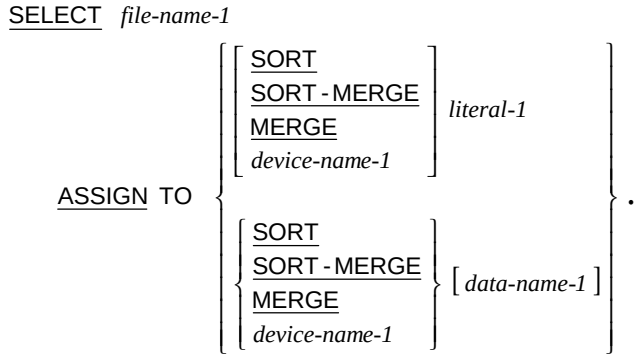
RESERVE Clause

The RESERVE clause allows the user to specify the number of input-output areas allocated. If the RESERVE clause is specified, the number of input-output areas allocated is equal to the value of *integer-1* if the ALTERNATE phrase is omitted or to the value of *integer-1* plus one if the ALTERNATE phrase is specified. The maximum number of input-output areas that can be allocated for a file is 255. Therefore, the maximum value that *integer-1* can have is 254 when the ALTERNATE phrase is specified, or 255 when the ALTERNATE phrase is not specified.

Specifying RESERVE NO ALTERNATE AREAS is the same as specifying RESERVE 1 AREA. Specifying RESERVE NO AREAS is the same as omitting the RESERVE clause. If the RESERVE clause is not specified, the number of input-output areas allocated defaults to a number appropriate for the runtime operating system. See the *RM/COBOL User's Guide* for more specific information on this point.

Sort-Merge File Control Entry

The sort-merge file control entry names a sort or merge file and specifies the association of the file to a storage-medium.



SELECT Clause

Each sort or merge file described in the Data Division must be described once and only once as a file-name in the FILE-CONTROL paragraph. Each sort or merge file specified in a file control entry must have a sort-merge file description entry in the Data Division. Since *file-name-1* represents a sort or merge file, only the ASSIGN clause is permitted to follow *file-name-1* in the FILE-CONTROL paragraph.

ASSIGN Clause

The ASSIGN clause specifies the association of the sort or merge file referenced by *file-name-1* to a storage medium (*device-name-1*), such as SORT, MERGE, SORT-MERGE or SORT-WORK.

The ASSIGN clause may also specify the file access name with *literal-1* or as the contents of a data item identified by *data-name-1*. If specified, the file access name must be correct both syntactically and semantically. However, for a sort-merge file, the value of the file access name is ignored by the object program.

If *literal-1* is specified, it must be a nonnumeric literal.

If *data-name-1* is specified, it must be defined in the Data Division as a data item of the category alphanumeric. *data-name-1* may be qualified.

I-O-CONTROL Paragraph

The I-O-CONTROL paragraph specifies the points at which rerun is to be established, the memory area which is to be shared by different files, and the location of files on a multiple file reel.

```
I-O-CONTROL. [
    [ rerun-entry ]...
    [ same-entry ]...
    [ multiple-file-entry ]... . ]
```

The I-O-CONTROL paragraph is optional. The clauses within the paragraph may appear in any order.

Any file-name referenced in the I-O-CONTROL paragraph must be specified in the FILE-CONTROL paragraph of the same program.

RERUN Clause

```
RERUN [ ON { file-name-1      } ]
           { rerun-name-1     }
      EVERY { { [ END OF ] { REEL   } } OF file-name-2 }
            { { integer-1  { UNIT   } }                }
            { { RECORDS                }               }
            { integer-2 CLOCK-UNITS                    }
            { condition-name-1                         }
```

The RERUN clause specifies when and where the rerun information is recorded. The RERUN clause, when specified, must satisfy the following rules:

1. *file-name-1* must be a sequentially organized file.
2. The END OF REEL or END OF UNIT phrase may be used only if *file-name-2* is a sequentially organized file.
3. When the END OF REEL or END OF UNIT phrase is used and *file-name-2* is not an output file, the ON phrase is required.

4. When either the *integer-1* RECORDS phrase or the *integer-2* CLOCK-UNITS phrase is specified, the ON phrase with *rerun-name-1* must be specified in the RERUN clause.

5. When *condition-name-1* is used, the ON phrase is required.

6. Only one RERUN clause containing the CLOCK-UNITS phrase may be specified.

7. *rerun-name-1* may be any user-defined word.

When either the END OF REEL or END OF UNIT phrase is used without the ON phrase, the rerun information is written on *file-name-2*, which must be an output file. When either the END OF REEL or END OF UNIT phrase is used and *file-name-1* is specified in the ON phrase, the rerun information is written on *file-name-1*, which must be an output file. In this case, *file-name-2* may be either an input or output file.

When the *integer-1* RECORDS phrase is used, the rerun information is written whenever *integer-1* records of *file-name-2* have been processed. *file-name-2* may be either an input or output file with any organization or access.

When the *integer-2* CLOCK-UNITS phrase is used, the rerun information is written whenever an interval of time, calculated by an internal clock, has elapsed.

When *condition-name* is used and *file-name-1* is specified in the ON phrase, the rerun information is written on *file-name-1*, which must be an output file, whenever a switch assumes a particular status as specified by the *condition-name-1*. The associated switch must be defined in the SPECIAL-NAMES paragraph.

More than one RERUN clause may be specified for a given *file-name-2*, provided that:

- When multiple *integer-1* RECORDS phrases are specified, no two of them may specify the same *file-name-2*.

- When multiple END OF REEL or END OF UNIT phrases are specified, no two of them may specify the same *file-name-2*.

SAME Clause

```
SAME [ RECORD     ]
     [ SORT       ]  AREA FOR  file-name-3 { file-name-4 }...
     [ SORT-MERGE ]
```

In the SAME clause, SORT and SORT-MERGE are equivalent.

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.

The file-names specified in a SAME clause may not reference an external file connector.

The four formats of the SAME clause (SAME AREA, 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. The following restrictions apply:

1. A file-name must not appear in more than one SAME AREA clause.
2. A file-name must not appear in more than one SAME RECORD AREA clause.
3. A file-name which represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.
4. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all the file-names in that SAME AREA clause must appear in 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. If a file-name that does not represent 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 that SAME AREA clause must be named in that SAME SORT AREA or SAME SORT-MERGE area clause.

The files referenced in the SAME AREA, SAME RECORD AREA, SAME SORT AREA, or SAME SORT-MERGE AREA clause need not all have the same organization or access.

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 storage areas assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time.

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 a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area (that is, records are aligned on the leftmost character position).

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. This clause specifies that storage is shared as follows:

1. The SAME SORT AREA or SAME SORT-MERGE AREA clause specifies a memory area which will be made available for use in sorting or merging each sort or merge file named. Thus any memory area allocated for the sorting or merging of a sort or merge file is available for reuse in sorting or merging any of the other sort or merge files.

2. In addition, storage areas assigned to files that do not represent sort or merge files may be allocated as needed for sorting or merging the sort or merge files named in the SAME SORT AREA or SAME SORT-MERGE AREA clause. In this implementation, no such sharing occurs during execution.

3. Files other than sort or merge files do not share the same storage area with each other. Users wishing these files to share the same storage area with each other must also include in the program a SAME AREA or SAME RECORD AREA clause naming these files.

4. During the execution of a SORT or MERGE statement that refers to a sort or merge file named in this clause, any non sort-merge files named in this clause must not be open.

MULTIPLE FILE Clause

```
MULTIPLE FILE TAPE CONTAINS

    { file-name-5 [ POSITION IS integer-3 ] }...
```

The MULTIPLE FILE clause is required when more than one file shares the same physical reel of tape and the operating system does not specify file positions. Regardless of the number of files on a single reel, only those files that are used in the object program need be specified. If all file-names have been listed in consecutive order, the POSITION clause need not be given.

If any file in the sequence is not listed, the position—one-relative to the beginning of the tape—must be specified in the POSITION clause. Whenever the POSITION clause is omitted, the position is assumed to be one greater than the position of the immediately preceding file in the MULTIPLE FILE clause, except for the first *file-name-5* which is assumed to be in position 1 when the POSITION clause is omitted.

The file-names specified in a MULTIPLE FILE clause may not reference an external file connector.

Not more than one file on the same tape reel may be open at one time.

Chapter 4: Data Division

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

The Data Division is optional. It is subdivided into five subordinate sections, each of which is itself optional. The entire Data Division may be omitted, but only when none of the subordinate sections are present.

Data Division Structure

The five subordinate sections in the Data Division are as follows:

1. The **File Section**, which 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. (See page 4-4.)

2. The **Working-Storage Section**, which describes records and noncontiguous data items which are not part of external data files but are developed and processed internally. It also describes data items whose values, assigned in the source program, do not change during execution of the object program. (See page 4-18.)

3. The **Linkage Section**, which describes formal arguments to be associated with actual arguments passed in the USING or GIVING phrases of a CALL statement and records to be based on a pointer value by use of the SET statement. (See page 4-19.)

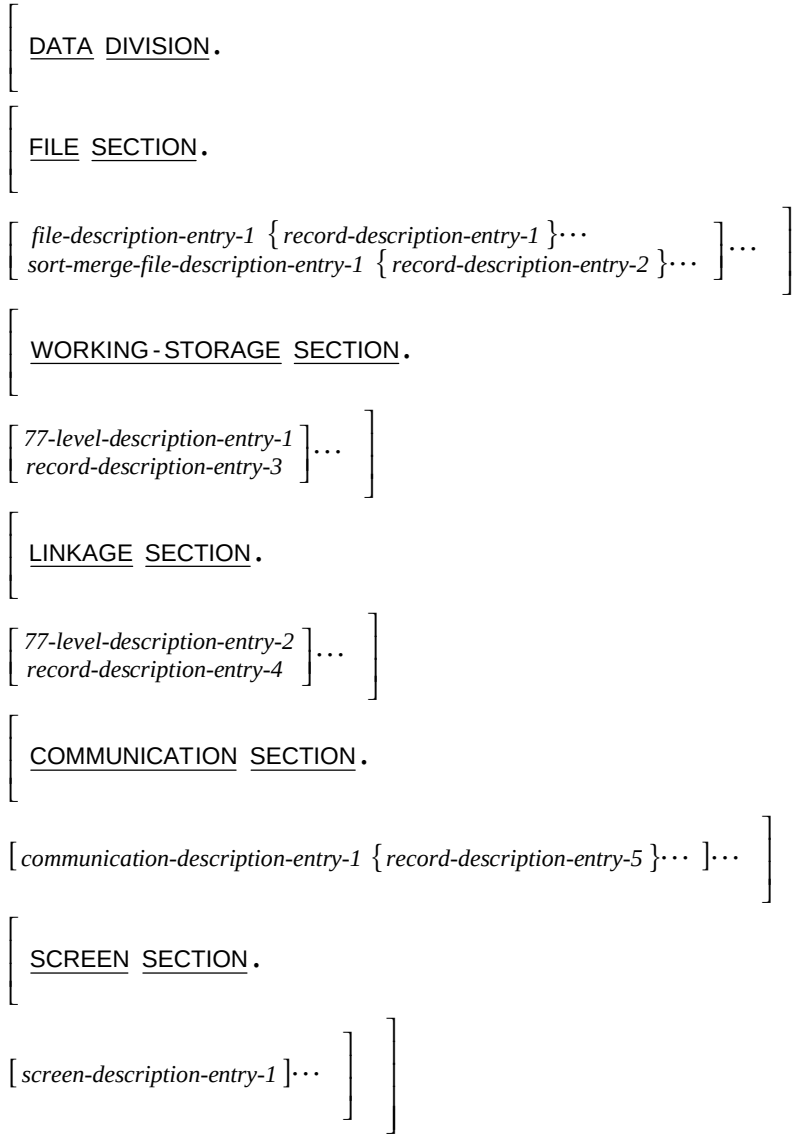
   No space is allocated in the program for data items defined in the Linkage Section of that program. Procedure Division references to these data items are resolved at runtime by replacing the reference in the program with the location assigned by the calling program for a formal argument associated with an actual argument or the location assigned by the most recently executed SET statement that established the base address for a based linkage record. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indexes for indexes defined in the Linkage Section.

   Data items defined in the Linkage Section of a program may be referenced within the Procedure Division of that program only if they are specified as operands of the USING or GIVING phrases of the Procedure Division header, or are subordinate to such operands, and the object program is under the control of a CALL statement that

specifies a USING or GIVING phrase that includes a corresponding actual argument to associate with the formal argument, or the SET statement has been used to associate an address with the linkage record. An exception to this rule is that the ADDRESS OF special register may reference the record-name and will return NULL if the reference requirements have not been satisfied.

4. The **Communication Section**, which describes the data items that serve as the interface between the Message Control System (MCS) and the program. (See page 4-22.)

5. The **Screen Section**, which describes the layout and attributes of fields on a terminal screen. It also provides for the automatic transfer of data between screen fields and data items defined in the other sections of the Data Division. (See page 4-22.)


DATA DIVISION.
FILE SECTION.
file-description-entry-1 { record-description-entry-1 }...
sort-merge-file-description-entry-1 { record-description-entry-2 }...
WORKING-STORAGE SECTION.
77-level-description-entry-1
record-description-entry-3
LINKAGE SECTION.
77-level-description-entry-2
record-description-entry-4
COMMUNICATION SECTION.
communication-description-entry-1 { record-description-entry-5 }...
SCREEN SECTION.
screen-description-entry-1

File Section

The File Section header is followed by file description entries or sort-merge file description entries consisting of a level indicator (FD and SD, respectively), a file-name and a series of independent clauses, terminated by a period. Each file description entry or sort-merge description entry is followed by one or more record description entries.

The file description entry and sort-merge file description entry (FD and SD) are the highest level of organization in the File Section.

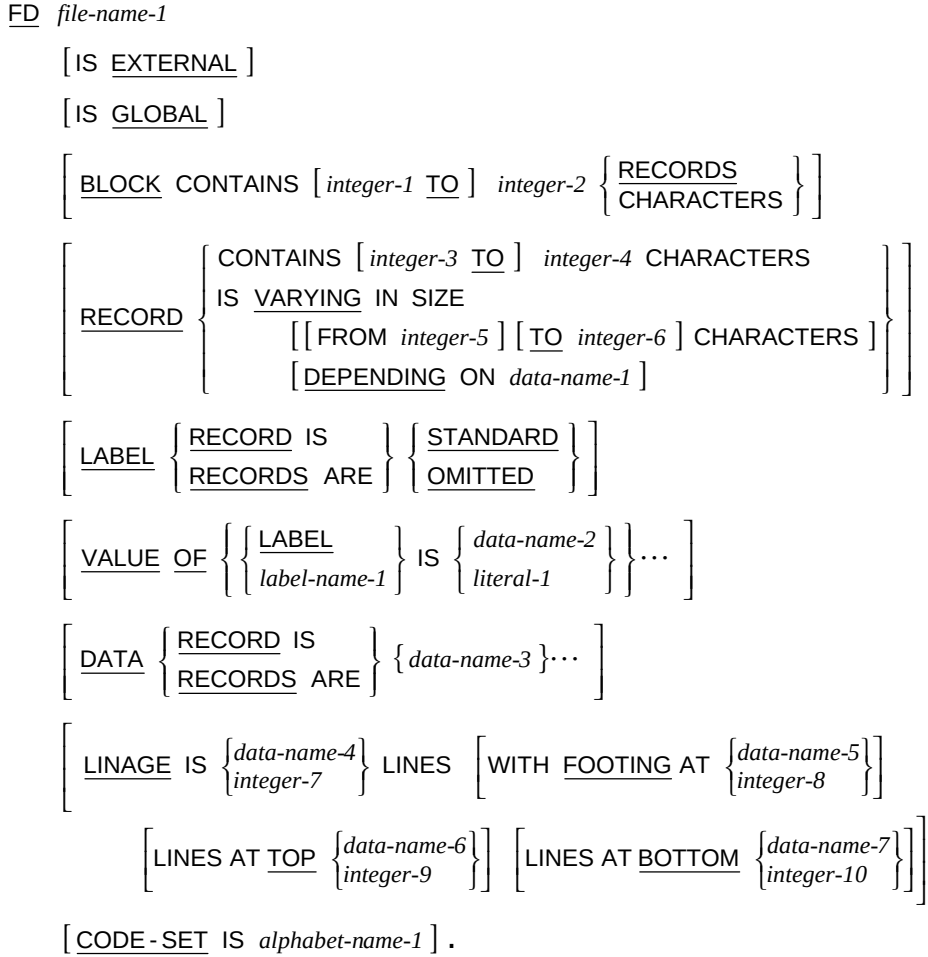
FILE SECTION.

[*file-description-entry-1* { *record-description-entry-1* }...]...
[*sort-merge-file-description-entry-1* { *record-description-entry-2* }...]

A Format 1 (data item initialization) VALUE clause specified in the File Section is ignored except in the execution of the INITIALIZE statement. The initial value of a data item in the File Section is undefined.

File Description Entry

The file description entry furnishes information concerning the physical structure, identification and record names pertaining to a given file.



The level indicator FD identifies the beginning of a file description and must precede the file-name.

The clauses that follow the name of the file are optional and their order of appearance is not significant.

The LINAGE clause may be used only if *file-name-1* references a sequential file. If the file description entry for a sequential file contains the LINAGE clause and the EXTERNAL clause, the LINAGE-COUNTER data item is an external data item. If the file description entry for a sequential file contains the LINAGE clause and the GLOBAL clause, the special register LINAGE-COUNTER is a global name.

One or more record description entries must follow the file description entry.

A file description entry must end with a period separator.

Sort-Merge File Description Entry

The sort-merge file description entry furnishes information concerning the physical structure, identification, and record-names of the file to be sorted or merged.

```
SD  file-name-1

[                 { CONTAINS [ integer-3 TO ]  integer-4 CHARACTERS                 }]
[                 { IS VARYING IN SIZE                                              }]
[ RECORD          {       [ [ FROM integer-5 ] [ TO integer-6 ] CHARACTERS ]        }]
[                 {       [ DEPENDING ON data-name-1 ]                              }]

[ DATA { RECORD IS   } { data-name-3 }··· ] .
[      { RECORDS ARE }                   ]
```

The level indicator SD identifies the beginning of the sort-merge file description and must precede the file-name.

The clauses that follow the name of the file are optional and their order of appearance is immaterial.

One or more record description entries must follow the sort-merge file description entry. Within the Procedure Division, *file-name-1* may not be used in OPEN, CLOSE, READ, START, DELETE or UNLOCK statements, nor may subordinate record-names be used in WRITE or REWRITE statements.

File Description Clauses

The file description clauses are placed between the FD or SD *file-name-1* declaration and the first of the subordinate record description entries. They serve to specify or document characteristics that are relevant to the file as a whole rather than to particular records.

BLOCK CONTAINS Clause

BLOCK CONTAINS [*integer-1* TO] *integer-2* { RECORDS / CHARACTERS }

The BLOCK CONTAINS clause specifies the size of a physical record.

This clause is required except when:

- A physical record contains only one complete logical record.
- The device assigned to the file has only one physical record size.
- The number of records contained in a block is specified in the runtime environment.

The size of the physical record may be stated in terms of RECORDS, unless one of the following situations exists, in which case the RECORDS phrase must not be used:

- In mass storage files where logical records may extend across physical records.
- The physical record contains padding.
- If logical records are grouped in such a manner that an inaccurate physical record size would be implied.

When the word CHARACTERS is specified, the physical record size is specified in terms of the number of character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record.

If only *integer-2* is specified, it represents the exact size of the physical record. If *integer-1* and *integer-2* are specified, they refer to the minimum and maximum size of the physical record, respectively.

If the associated file connector is an external file connector, all BLOCK CONTAINS clauses in the run unit which are associated with that file connector must have the same value for *integer-1* and *integer-2*.

CODE-SET Clause

CODE-SET IS *alphabet-name-1*

The CODE-SET clause specifies the character code convention used to represent data on the external media.

When the CODE-SET clause is specified for a file, all data in that file must be described as usage is DISPLAY and any signed numeric data must be described with the SIGN IS SEPARATE clause.

If the CODE-SET clause is specified, *alphabet-name-1* specifies the character code convention used to represent data on the external media. It also specifies the algorithm for converting the character codes on the external media to or from the native character codes. This code conversion occurs during the execution of an input or output operation. (See the discussion of the SPECIAL-NAMES paragraph that begins on page 3-7.)

If the CODE-SET clause is not specified, the native character code set is assumed for data on the external media.

If the CODE-SET clause is specified in both the file control entry and the file description entry for a file, the two alphabet-names must be the same.

If the associated file connector is an external file connector, all CODE-SET clauses in the run unit which are associated with that file connector must have the same character set.

DATA RECORDS Clause

DATA { RECORD IS / RECORDS ARE } { *data-name-3* }...

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

data-name-3 is the name of a data record and must have a 01 level-number record description, with the same name, associated with it.

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 different sizes, different formats, and so forth. The order in which they are listed is not significant.

All data records within a file share the same area, whether or not they are of the same type.

EXTERNAL Clause

```
IS EXTERNAL
```

The EXTERNAL clause specifies that a file connector is external.

Use of the EXTERNAL clause does not imply that the associated file-name is a global name.

The file connector associated with this description entry is an external file connector. The data records described subordinate to this file description entry, as well as any data items described subordinate to the data description entries for such records, attain the external attribute.

GLOBAL Clause

```
IS GLOBAL
```

The GLOBAL clause specifies that a file-name is a global name. A global name is available to every program contained within the program which declares it.

A file-name described using a GLOBAL clause is a global name. All *data-names* subordinate to a global name are global names. All *condition-names* and *split-key-names* associated with a global name are global names.

A statement in a program contained directly or indirectly within a program which describes a global name may reference that name without describing it again.

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.

LABEL RECORDS Clause

```
LABEL { RECORD IS  } { STANDARD }
      { RECORDS ARE } { OMITTED  }
```

The LABEL RECORDS clause specifies whether labels are present.

The OMITTED option specifies that no explicit labels exist for the file or the device to which the file is assigned. The STANDARD option specifies that labels exist for the file or the device to which the file is assigned and that they conform to the conventions of the runtime environment. See the *RM/COBOL User's Guide* for more information.

Omission of the LABEL RECORDS clause from a file description entry is equivalent to specifying LABEL RECORDS OMITTED.

If a VALUE OF clause is present in a file description entry, a LABEL RECORDS OMITTED clause is not allowed.

If the file connector associated with this file description entry is an external file connector, all LABEL RECORDS clauses in the run unit which are associated with this file connector must have the same specification.

LINAGE Clause

```
LINAGE IS { data-name-4 } LINES [ WITH FOOTING AT { data-name-5 } ]
          { integer-7   }       [                 { integer-8   } ]
    [ LINES AT TOP { data-name-6 } ] [ LINES AT BOTTOM { data-name-7 } ]
                   { integer-9   }                     { integer-10  }
```

The LINAGE clause provides a means for specifying the depth of a logical page in number of lines. It also allows for the specification of the top and bottom margins on the logical page and the line number at which the footing area begins.

data-name-4, *data-name-5*, *data-name-6*, *data-name-7* must reference unsigned numeric integer data items.

data-name-4, *data-name-5*, *data-name-6*, *data-name-7* may be qualified.

The LINAGE clause may only be used in a file description entry for a sequential organization file.

The LINAGE clause provides a means for specifying the size of a logical page in terms of number of lines. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the values of these items are zero. If the FOOTING phrase is not specified, no end-of-page condition independent of the page overflow condition exists.

There is not necessarily any relationship between the size of the logical page and the size of the physical page. Each logical page is contiguous to the next with no additional spacing provided. When a LINAGE file is written, form feed characters are not used because they cause the printer to advance to the next physical page. The LINAGE-PAGES-PER-PHYSICAL-PAGE in the PRINT-ATTR runtime configuration record may be used to cause form feeds to be generated between a specified number of logical pages, that is, the option specifies the number of logical pages that fit on a physical page.

integer-7 or the value of the data item referenced by *data-name-4* specifies the number of lines that can be written, spaced, or both, on the logical page. The value must be greater than zero. That part of the logical page in which these lines can be written or spaced is called the page body.

integer-8 or the value of the data item referenced by *data-name-5* 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 *integer-7* or the value of the data item referenced by *data-name-4*.

The footing area comprises the area of the page body between the line represented by *integer-8* or the value of the data item referenced by *data-name-5*, and the line represented by *integer-7* or the value of the data item referenced by *data-name-4*, inclusive. When lines are written or spaced in the footing area, an end-of-page condition occurs. The end-of-page condition can be detected by the END-OF-PAGE (or EOP) phrase of the WRITE statement.

integer-9 or the value of the data item referenced by *data-name-6* specifies the number of lines that comprise the top margin on the logical page. The value may be zero.

integer-10 or the value of the data item referenced by *data-name-7* specifies the number of lines that comprise the bottom margin on the logical page. The value may be zero.

integer-7, *integer-9* and *integer-10*, if specified, are 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 make up each of the indicated sections of a logical page. *integer-8*, if specified, is used at that time to define the footing area. These values are used for all logical pages written for that file during an execution of the program.

The values of the data items referenced by *data-name-4*, *data-name-6* and *data-name-7*, if specified, are used as follows:

- The values of the data items, at the time an OPEN statement with the OUTPUT phrase is executed for the file, are used to specify the number of lines that make up each of the indicated sections for the first logical page.
- The values of the data items, at the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, are used to specify the number of lines that make up each of the indicated sections for the next logical page.

The value of the data item referenced by *data-name-5*, if specified, at the time an OPEN statement with the OUTPUT phrase is executed for the file, is 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 is used to define the footing area for the next logical page.

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:

1. A separate LINAGE-COUNTER is supplied for each file whose file description entry contains a LINAGE clause.

2. LINAGE-COUNTER may be referenced only in Procedure Division statements; however, only the runtime system may change the value of LINAGE-COUNTER. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by *file-name-1* when necessary.

3. The LINAGE-COUNTER special register behaves as if it were described as PIC 9(*n*) BINARY, where *n* represents the number of 9's in the PICTURE character-string for *data-name-4* or the number of digits specified in *integer-7*. The number of character positions (bytes) allocated for the LINAGE-COUNTER special register is determined by the value of *n* and the configured binary allocation scheme.

If the file description entry for a sequential file contains the LINAGE clause and the EXTERNAL clause, the LINAGE-COUNTER data item is an external data item. If the file description entry for a sequential file contains the LINAGE clause and the GLOBAL clause, the special register LINAGE-COUNTER is a global name.

LINAGE-COUNTER is automatically modified, according to the following rules, during the execution of a WRITE statement to an associated file:

1. When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to one. During the resetting of the LINAGE-COUNTER to the value one, the value of LINAGE-COUNTER is implicitly incremented to exceed the value specified by *integer-7* or the data item referenced by *data-name-4.*

2. When the ADVANCING *identifier-2* or *integer-1* phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by *integer-1* or the value of the data item referenced by *identifier-2.*

3. When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value one.

4. The value of LINAGE-COUNTER is automatically reset to one when the device is repositioned to the first line that can be written on for each of the succeeding logical pages.

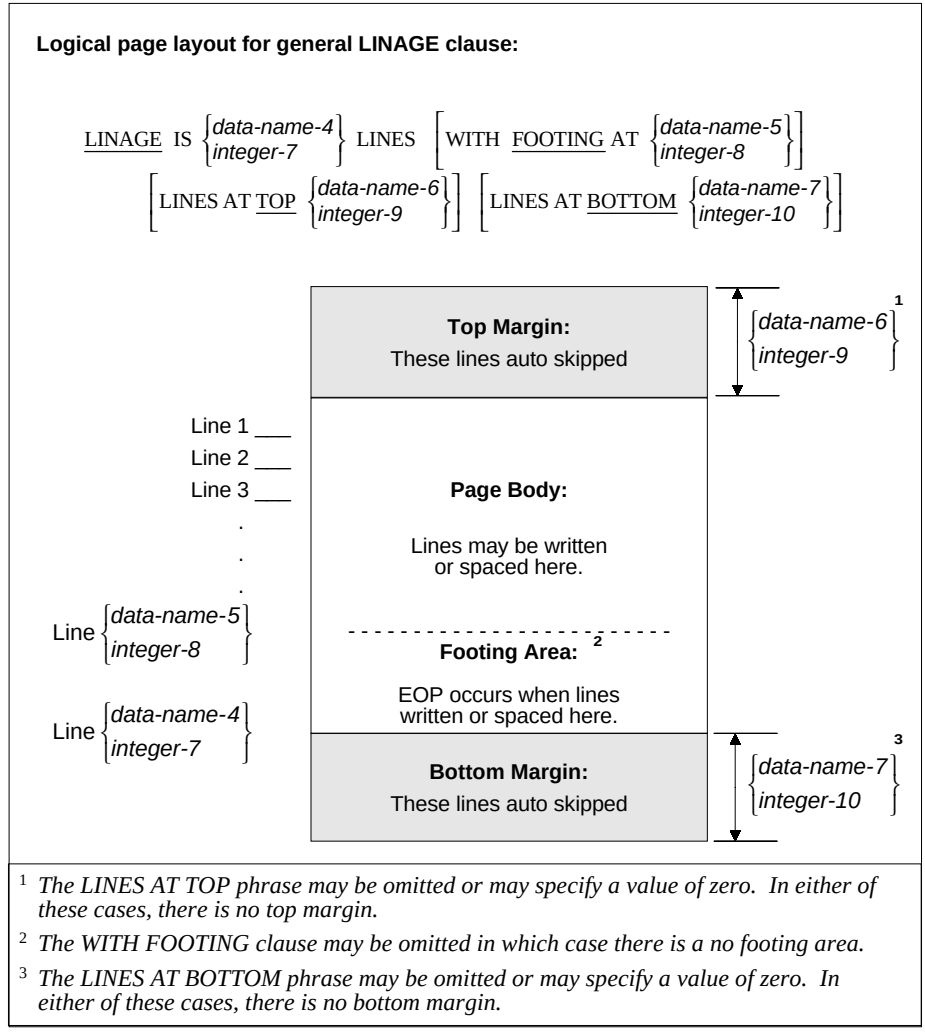
   The value of LINAGE-COUNTER is automatically set to one at the time an OPEN statement with the OUTPUT phrase is executed for the associated file.

If the file connector associated with this file description entry is an external file connector, all file description entries in the run unit which are associated with this file connector must have:

1. A LINAGE clause, if any file description entry has a LINAGE clause.
2. The same corresponding values for *integer-7*, *integer-8*, *integer-9* and *integer-10*, if specified.
3. The same corresponding external data items referenced by *data-name-4*, *data-name-5*, *data-name-6* and *data-name-7*.

Figure 4-1 and Figure 4-2 illustrate the logical page layout for the LINAGE clause.

Figure 4-1: General LINAGE Clause

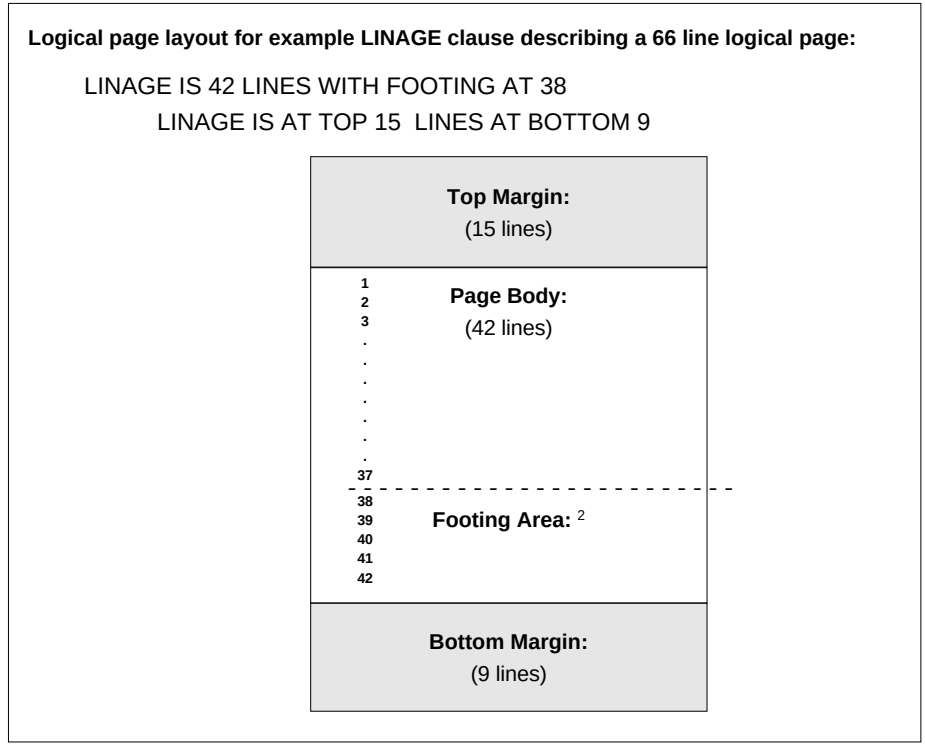


[1] *The LINES AT TOP phrase may be omitted or may specify a value of zero. In either of these cases, there is no top margin.*

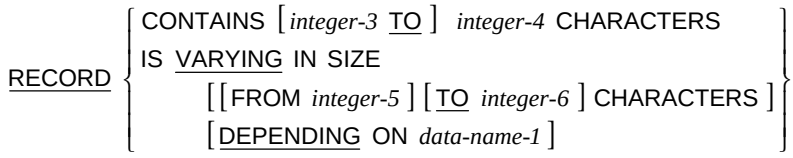
[2] *The WITH FOOTING clause may be omitted in which case there is a no footing area.*

[3] *The LINES AT BOTTOM phrase may be omitted or may specify a value of zero. In either of these cases, there is no bottom margin.*

Figure 4-2: Example LINAGE Clause



RECORD Clause



The RECORD clause specifies the size of the data records.

Record descriptions for the file must not describe records which contain less character positions than that specified by *integer-3*, *integer-5* or records which contain more character positions than that specified by *integer-4* or *integer-6*.

integer-4 must be greater than or equal to *integer-3*.

integer-6 must be greater than *integer-5*.

data-name-1 must describe an elementary unsigned integer in the Working-Storage or Linkage Section.

If the RECORD clause is not specified, the size of each data record is fully defined in the record description entry. If all record description entries describe the same number of character positions—and none contain Format 2 of the OCCURS clause—the file will be a fixed-length record file; otherwise, the file will be a variable-length record file.

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-3* and *integer-4*, or *integer-5* and *integer-6*. If the RECORD clause is not specified, all record description entries associated with this file connector must be the same length.

1. *integer-4*, used by itself, indicates that all the data records in the file have the same size. In this case, *integer-4* represents the exact number of characters in the data record. The file will be a fixed-length record file, even if varying length record descriptions are associated with it.

2. If *integer-3* and *integer-4* are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively. If *integer-3* is not equal to *integer-4*, the file will be a variable-length record file, even if fixed-length record descriptions are associated with it.

3. The size 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 sum of the number of characters in all fixed-length elementary items, plus any filler characters generated between elementary items because of explicit or implicit synchronization. If the record is variable length, the minimum number of characters in a variable-occurrence data item is added to the fixed size to get the minimum record size. The maximum number is added to the fixed size to get the maximum record size.

The IS VARYING IN SIZE phrase is used to specify variable record lengths. *integer-5* specifies the minimum number of character positions in any record of the file. *integer-6* specifies the maximum number of character positions in any record in the file.

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.

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*.

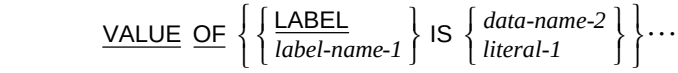
During the execution of a RELEASE, REWRITE or WRITE statement, the number of character positions in the record is determined by one of the following conditions:

- If *data-name-1* is specified, by the content of the data item referenced by *data-name-1*.
- If *data-name-1* is not specified and the record does not contain a variable-occurrence data item, by the number of the character positions in the record.
- If *data-name-1* is not specified and the record contains a variable-occurrence data item, by the sum of the fixed portion and that portion of the table described by the number of occurrences at the time of the execution of the output statement.

If *data-name-1* is specified, after the successful execution of a READ or RETURN statement for the file, the contents of the data item referenced by *data-name-1* will indicate the number of character positions in the record just read or returned.

When an INTO phrase is specified in a READ or RETURN statement, the number of character positions in the current record that participate as the sending data item in the implied MOVE is the number of character positions in the record just read or returned.

VALUE OF Clause



The VALUE OF clause particularizes the description of an item in the label records associated with a file or specifies the file access name.

label-name-1 may be FILE-ID, LABEL or any user-defined word.

When *label-name-1* is FILE-ID, *data-name-2* or *literal-1* specifies the file access name for the file. VALUE OF FILE-ID provides an alternative to specifying the file access name in the ASSIGN clause of the file control entry. If the file access name is specified in both alternatives, the same data-name or literal must be specified in each; otherwise, the value specified in the file control entry will take precedence. If the file access name is not specified in either the file control entry or the file description entry, then *file-name-1* is used as the file access name. The value of the file access name, however specified, must be valid according to the requirements of the runtime input-output system. If *data-name-2* is specified for the file access name, at the time of an OPEN statement

execution for *file-name-1*, the value of the data item to which *data-name-2* refers will be used as the file access name.

When *label-name-1* is LABEL, *data-name-2* or *literal-1* particularizes the description of an item in the label records associated with the file. The value of this data item or literal is available to the runtime input-output system, but is not currently used for any purpose. LABEL must not be specified for *label-name-1* when the OMITTED option is specified in the LABEL RECORDS clause.

When *label-name-1* is a user-defined, the phrase is treated as commentary. *data-name-2* or *literal-1* must by syntactically correct, but have no effect on the object program.

data-name-2 may be qualified. *data-name-2* must be defined in the Working-Storage Section and must not be described with the USAGE IS INDEX clause.

A figurative constant may be substituted for *literal-1*.

If the associated file connector is an external file connector, all VALUE OF clauses in the run unit which are associated with that file connector must be consistent.

Working-Storage Section

The Working-Storage Section is made up of the section header, followed by data description entries for 77-level description entries, record description entries, or both.

A data-name defined at the 01 or 77 level in the Working-Storage Section must be unique only if there is a reference to it elsewhere in the program. Subordinate data-names need not be unique if they can be made unique by qualification or if there are no references to them elsewhere in the program.

WORKING-STORAGE SECTION.

[*77-level-description-entry-1*]
[*record-description-entry-3*] ...

Linkage Section

The structure of the Linkage Section is identical to the Working-Storage Section. That is, it consists of a section header, followed by data description entries for noncontiguous data items, record description entries, or both.

A data-name defined at the 01 or 77 level in the Linkage Section must be unique only if there is a reference to it elsewhere in the program. Subordinate data-names need not be unique if they can be made unique by qualification or if there are no references to them elsewhere in the program.

LINKAGE SECTION.

[*77-level-description-entry-2*]
[*record-description-entry-4*] ...

Record description entries and 77-level-description-entries in the Linkage Section describe record layouts for formal arguments of a program and for based linkage records. Linkage Section data items are not allocated storage during compilation, but rather during the execution of the run unit.

The formal arguments of a program are named in the USING and GIVING phrases of the Procedure Division header and must be names defined as level 01 or level 77 entries in the Linkage Section. Formal arguments receive their base address from the actual arguments passed by a calling program. Formal arguments may also be treated as based linkage records; this can be convenient to establish a default argument when the calling program does not pass the corresponding actual argument.

Based linkage data records are any record-description-entries or 77-level-description-entries in the Linkage Section that receive their base address by use of Formats 5 or 6 of the SET statement in which the receiving item is an ADDRESS OF *data-name-1*. Based linkage records may include formal arguments of a program. For example, it may be convenient to set the base address of a formal argument when the corresponding actual argument is omitted.

When a program is placed into its initial state (either on its first CALL in the run unit or on its first CALL since it has been canceled), the base addresses of all based linkage records are set equal to NULL. A Format 5 SET statement must be executed to change the base address to a value other than NULL. Once set, the base address of a based linkage record remains set until changed by the execution of a Format 5 or Format 6 SET statement or the program that describes the based linkage item is canceled. If the program refers to a data item with a NULL base address, other than in an ADDRESS OF special register or in the USING or GIVING phrases of a CALL statement, a runtime data

reference error will terminate the run unit. The ADDRESS OF special register may be used to test for a base address that is equal to NULL.

If a based linkage item is also a formal argument, the actual argument base address in a subsequent CALL statement in the calling program overrides any base address set or modified by a Format 5 or Format 6 SET statement in the called program. The override occurs each time that the program is called, unless the actual argument base address is equal to NULL. When the actual argument base address is equal to NULL, the last set base address is used instead. If there has been no Format 5 SET executed to set the base address, the NULL address from the initial state of the program will be used for a reference and a data reference error will occur except as described in the preceding paragraph. An actual argument has a NULL base address in the following cases:

- The actual argument has been omitted from the CALL statement in the calling program, either by specifying fewer arguments than the number of expected formal arguments in the called program or by specifying OMITTED for the actual argument in the calling program.

- The actual argument was specified in the CALL statement in the calling program, but is a formal argument or based linkage record that has a NULL base address. Note that a pointer data item that has a NULL value is **not** the same as a based linkage item with a NULL base address. That is, passing a pointer data item as an actual argument passes the base address of the pointer data item and it is the data item value that is NULL, not the base address.

The ENTRY-LINKAGE-SETTINGS keyword of the COMPILER-OPTIONS configuration record may be used to control certain details of how base addresses for linkage records are interpreted at runtime for a program compiled with a particular setting of this keyword. The option controls what happens on each entry to a called program, including how the correspondence of actual arguments to formal arguments and previous executions of Format 5 and 6 SET statements affect the base address used for a reference to a Linkage Section data item during that invocation of the called program. See Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, for an explanation of this compiler configuration option.

A Format 1 (data item initialization) VALUE clause specified in the Linkage Section is ignored except in the execution of the INITIALIZE statement. If the runtime element containing the Linkage Section is activated by a COBOL runtime element, the initial value of a data item in the Linkage Section is determined by the value of the corresponding formal parameter in the activating runtime element, as described in the paragraphs above and the general rules of the Procedure Division, which begin on page 5-1. If the runtime element containing the Linkage Section is activated by the operating system, the initial value of a Linkage Section data item is as described in Chapter 7, *Running*, of the *RM/COBOL User's Guide*.

The compiler handles as a special case the specification of a Linkage Section record-name as an actual argument in a CALL statement or in a reference modified identifier. In these two cases, the record-name is resolved according to the description of the actual data item on which the record-name is based rather than using the Linkage Section description of the record-name. The record-name is based on an actual argument if it represents a formal argument, that is, is named in the Procedure Division header USING or GIVING argument list, or may be based on some other data item through use of Formats 5 and 6 of the SET statement. Other than when used as an actual argument or in a reference modified identifier, a Linkage Section record-name is resolved according to its data description entry in the Linkage Section of the program in which it is declared.

This special case means that a program that is just an intermediary between two programs need not have a Linkage Section data description entry that accurately describes the size of the actual argument being passed through it. For example, calling C$CARG with a formal argument, which is described as longer than the corresponding actual argument, will no longer result in a data reference error. Instead, C$CARG will return the correct length of the actual argument, and because of the reference modification change described here, this length may be successfully used to reference modify the formal argument in order to access the entire contents of the actual argument. This also means that a program can call the supplied subprogram C$CARG with an argument that the calling program omitted without getting a data reference error. In this case, the call to C$CARG will succeed and return an argument descriptor that includes a type of OMITTED and a length of zero.

In the case of reference modification, an omitted actual argument would cause a data reference error, but for an argument that is not omitted, the reference modification can use any offset and length combination that is consistent with the actual argument. Previous to this enhancement, reference modification that used variables implied a reference to the item as described in the Linkage Section for the formal argument data item and this implied reference, if larger than the corresponding actual argument, would cause a data reference error before the reference modification was applied.

This special case also means that when the supplied subprogram, C$MemoryAllocate, is used to allocate an area of memory and then the SET statement is used to base a Linkage Section record on this allocated memory, the entire allocated memory area is passed as an actual argument when the record-name is used in the USING or GIVING phrases of the CALL statement. Also, the entire allocated memory area may be accessed by using reference modification of the record-name.

Communication Section

The Communication Section is made up of the section header, followed by communication description entries consisting of a level indicator (CD), a cd-name and a series of independent clauses. The communication description entry is terminated by a period.

The record-description entry associated with the Communication Section may be implicitly redefined by user-specified record description entries written immediately following the communication description entry.

COMMUNICATION SECTION.
[*communication-description-entry-1* { *record-description-entry-5* }...]...

Screen Section

The syntactic structure of the Screen Section resembles that of the Working-Storage Section. That is, it consists of a section header followed by zero, one, or more entries, each of which consists of a required level number followed by a series of optional clauses.

The entries specify the appearance of a rectangular display area called a screen. The maximum meaningful horizontal and vertical dimensions of the screen are determined by the hardware characteristics of the terminal associated with the run unit. The common limit for the horizontal dimension is 80 character positions, and the common limit for the vertical dimension is 25 lines.

Screen entries may be used to define all or any portion of the physical screen, and the entire screen or any subregion of it may be redefined as many times as is needed by the program.

Level numbers are used in the same way as in the other sections of the Data Division. That is, level 77 entries are used to describe screen items not part of a larger structure, and not subdivided into subordinate entries. Level numbers 01 through 49 can be used to define screen entries that are organized in a hierarchical structure: level 01 is the most inclusive. Level numbers 66 and 88 may not be used in the Screen Section.

Each entry in the Screen Section may define a screen-name. The rules regarding uniqueness of screen-names are the same as the rules regarding uniqueness of data-names in the other sections of the Data Division. That is, a screen-name defined at the 01 or 77 level in the Screen Section must be unique only if there is a reference to it elsewhere in the program. Subordinate screen-names (those at level numbers 02 through 49) need not

be unique if they can be made unique by qualification or if there are no references to them elsewhere in the program.

Screen-names defined in the Screen Section do not represent data items, and they can be referred to elsewhere in the program only in ACCEPT and DISPLAY statements, as explained beginning on pages 6-2 and 6-58, respectively.

SCREEN SECTION.
[*screen-description-entry-1*]...

Record Description Entry

A record description entry 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 and a series of independent clauses, as required.

{ *data-description-entry-1* }...

Level-Numbers

The first data description of a record must have a level-number of 01 or 1, and must start in area A of a source line.

Any data item whose description specifies a level-number in the range 01 through 48 may be subdivided into one or more subordinate data items. When this is done, the subdivided data item becomes a group item. The subdivision is accomplished by following the data description of the group item by one or more further data item descriptions, each having the same level-number. The common level-number selected for these immediately subordinate data items must be larger (by one or more) than the level-number of the group data item but less than 50.

Each subordinate data item may in turn be subdivided by the same process, and the nesting of subordinates within subordinates is limited only by the availability of increasing level-numbers that are less than 50. This arrangement of data definitions results in a hierarchical data structure. The rank of the constituent data items is determined by the numerical value of its level-number: the smaller the level-number, the more inclusive the data item and the higher its rank.

Elementary Items

Any data description entry which is not further subdivided is called an elementary item. A record itself may be an elementary item, consisting of a single level-01 data description entry. A subdivided data description entry with its subdivisions is called a group and is nonelementary. Therefore, 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.

Note that certain clauses of the data description entry may occur only in elementary items. They may not occur in a nonelementary entry as they may affect the subdivisions of that entry. The description of an elementary item must have either a PICTURE clause or INDEX usage; it may not have both.

77-Level Description Entry

In the Working-Storage and Linkage Sections, a special level-number of 77 can be used in data description entries that are not subdivisions of other items, and are not themselves subdivided. These data description entries specify noncontiguous data items. Such a data description entry is elementary.

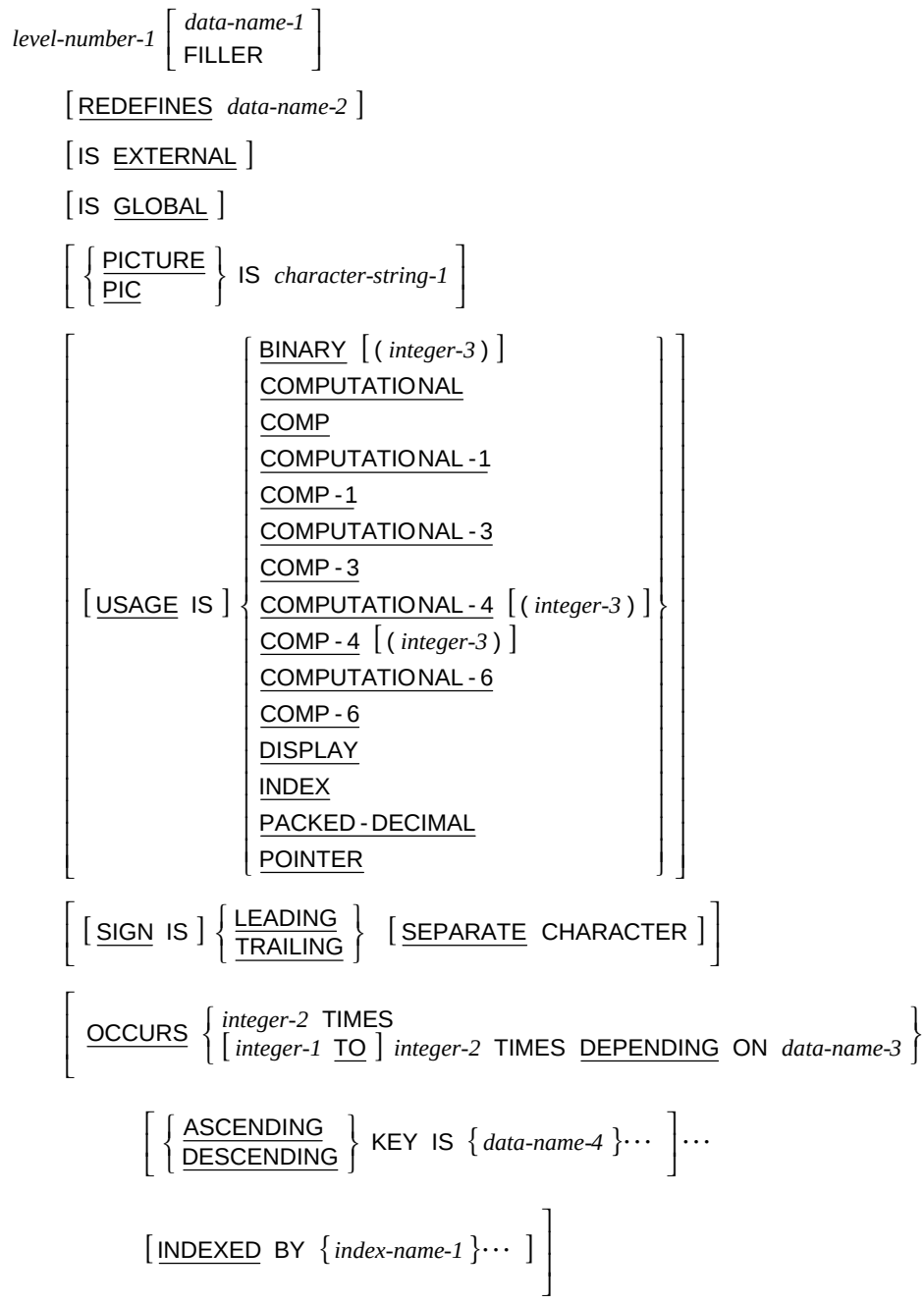
A 77-level data description entry must contain a data-name and either the PICTURE clause or the USAGE IS INDEX clause, but cannot contain an OCCURS clause. Other clauses are optional and can be used to complete the description of the item if necessary.

data-description-entry-2

Data Description Entry

A data description entry specifies data item characteristics.

Format 1: Data-Name Full Declaration



Format 1: Data-Name Full Declaration *(continued from previous page)*

```
[ { SYNCHRONIZED } [ LEFT  ] ]
  { SYNC         } [ RIGHT ] ]

[ { JUSTIFIED } RIGHT ]
  { JUST      }

[ BLANK  WHEN  ZERO ]

[ VALUE  IS  literal-1 ] .
```

Format 2: Data-Name Renames

```
66  data-name-1

    RENAMES  data-name-2  [ { THROUGH } data-name-3 ] .
                            { THRU    }
```

Format 3: Condition-Name Declaration

```
88  condition-name-1

                                  { literal-1 [ { THROUGH } literal-2 ] }
    { VALUE  IS     }             {             { THRU    }           ]  }…
    { VALUES  ARE   }             {                                      }
                                  {                                      }
                                  { relational-operator  literal-1       }

        [ WHEN  SET  TO  FALSE  IS  literal-3 ] .
```

Format 4: Constant-Name Declaration

```
78  constant-name-1

    VALUE  IS { literal-1            } .
              { constant-expression-1 }
```

The clauses may be written in any order except that *data-name-1* or the FILLER clause, if specified, must immediately follow *level-number*.

The PICTURE clause must not be specified for the subject of a RENAMES clause or for an item whose usage is index or pointer. For any other entry describing an elementary

item, a PICTURE clause must be specified except that the PICTURE clause may be omitted for an elementary item when a nonnumeric literal is specified in the VALUE clause. In the latter case, a PICTURE clause is implied of the form 'PICTURE X(*length*)', where *length* is the length of the literal specified in the VALUE clause.

The words THRU and THROUGH are equivalent.

The clauses SYNCHRONIZED, PICTURE, JUSTIFIED and BLANK WHEN ZERO, must not be specified except for an elementary data item.

The EXTERNAL clause may be specified only in data description entries in the Working-Storage Section whose level-number is 01.

The EXTERNAL clause and the REDEFINES clause must not be specified in the same data description entry.

The GLOBAL clause may be specified only in data description entries whose level-number is 01.

data-name-1 must be specified for any entry containing the GLOBAL or EXTERNAL clause, or for record descriptions associated with a file description entry which contains the EXTERNAL or GLOBAL clause.

Format 3 is used to define 88-level condition-names. 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 follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry which contains a level-number except the following:

- Another 88-level condition-name
- A level 66 item (RENAMES)
- A level 78 item (constant-name)
- A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY)
- An index data item

Format 4 is used to define constant-names. Each constant-name requires a separate entry with level-number 78. Format 4 contains the name of the constant and the value associated with the constant-name. The constant-name may be used wherever a literal is specified in a format, unless otherwise forbidden. The effect of specifying a constant-name is as if the literal value associated with the constant-name had been specified instead of the constant-name. A constant-name with an integer value may also

be used wherever an integer value is specified in a format or as the repeat count in a PICTURE character-string. A constant-name with an integer value may also be used as a level-number or segment-number.

A constant-name may only be used after it has been declared in a data description entry. That is, a constant-name must not be the object of a forward reference.

A constant-name may not be used for a literal text-name or literal library-name in a COPY statement, or a literal program-name in a PROGRAM-ID paragraph or END PROGRAM header.

Constant-names are implicitly global.

Each data description entry must end with a period separator.

BLANK WHEN ZERO Clause

BLANK WHEN ZERO

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

The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric or numeric edited and whose usage is explicitly or implicitly DISPLAY.

The BLANK WHEN ZERO clause must not be specified in the same entry with a PICTURE clause having an asterisk as the zero suppression symbol.

The BLANK WHEN ZERO clause must not be specified in the same entry with a PICTURE clause that specifies an operational sign with the symbol S. However, if the separate sign option is specified in the Compile Command, then the BLANK WHEN ZERO clause may be specified in the same entry with a PICTURE clause that specifies an operational sign; in this case, the operational sign symbol S is ignored and a trailing symbol + assumed in the PICTURE character-string.

When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero.

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.

Data-Name or FILLER Clause

[*data-name-1*]
[FILLER]

A data-name specifies the name of the data being described. The keyword FILLER specifies an item of the logical record that cannot be referred to explicitly.

If either *data-name-1* or the keyword FILLER is specified, it must be the first word following the level-number in each data description entry. If this clause is omitted, the data item being described is treated as though FILLER had been specified.

The keyword FILLER may be used to name a data item. Under no circumstances can a FILLER item be referred to explicitly. However, the keyword FILLER may be used to name a conditional variable: such use does not require explicit reference to the FILLER item, but to its value.

EXTERNAL Clause

IS EXTERNAL

The EXTERNAL clause specifies that a data item is external. The constituent data items and group data items of an external data record are available to every program in the run unit which describes that record.

The EXTERNAL clause may be specified in record description entries in the Working-Storage Section.

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

The VALUE clause must not be used in any data description entry which includes, or is subordinate to, an entry which includes the EXTERNAL clause. The VALUE clause may be specified for condition-name entries associated with such data description entries.

The data contained in the record named by the data-name clause is external and may be accessed and processed by any program in the run unit which describes and, optionally, redefines it subject to the rules set forth in the paragraphs that follow.

Within a run unit, if two or more programs describe the same external data record, each record-name of the associated record description entries must be the same and the records must define the same number of standard data format characters. However, a program which describes an external record may contain a data description entry including the

REDEFINES clause which redefines the complete external record, and this complete redefinition need not occur identically in other programs in the run unit.

Use of the EXTERNAL clause does not imply that the associated data-name is a global name.

GLOBAL Clause

IS GLOBAL

The GLOBAL clause specifies that a data-name is a global name. A global name is available to every program contained within the program which declares it.

The GLOBAL clause may be specified in record description entries in the File Section or the Working-Storage Section.

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

A data-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.

A statement in a program contained directly or indirectly within a program which describes a global name may reference that name without describing it again.

If the GLOBAL clause is used in a data description entry which contains the REDEFINES clause, it is only the subject of that REDEFINES clause which possesses the global attribute.

JUSTIFIED Clause

{ JUSTIFIED / JUST } RIGHT

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

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 a 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.

When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply.

The JUSTIFIED clause cannot be specified for an index data item or for any data item described as numeric or for which editing is specified.

The JUSTIFIED clause can be specified only at the elementary item level.

JUST is an abbreviation for JUSTIFIED.

Level-Number

level-number-1

The level-number shows the hierarchy of data within a logical record. In addition, it identifies entries for working storage items, linkage items, condition-names, constant-names, and the RENAMES clause.

level-number-1 is required as the first element in each data description entry.

Data description entries subordinate to a CD, FD or SD entry must have level-numbers with values 01 through 49, 66, 78, or 88.

Data description entries in the Working-Storage Section and Linkage Section must have level-numbers with the values 01 through 49, 66, 77, 78, or 88.

The level-number 01 identifies the first entry in each record description.

Level-number 66 is assigned to identify RENAMES entries.

Level-number 77 is assigned to identify noncontiguous working storage data items and noncontiguous linkage data items.

Level-number 78 is assigned to identify constant-names.

Level-number 88 is assigned to identify condition-names associated with a conditional variable.

Multiple level 01 entries subordinate to any given level indicator CD, FD or SD, represent implicit redefinitions of the same area.

OCCURS Clause

Format 1: Fixed Number of Occurrences

```
OCCURS integer-2 TIMES

    [ { ASCENDING  } KEY IS { data-name-4 }... ]...
      { DESCENDING }

    [INDEXED BY { index-name-1 }... ]
```

Format 2: Variable Number of Occurrences

```
OCCURS [ integer-1 TO ] integer-2 TIMES DEPENDING ON data-name-3

    [ { ASCENDING  } KEY IS { data-name-4 }... ]...
      { DESCENDING }

    [INDEXED BY { index-name-1 }... ]
```

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts.

The OCCURS clause is used in defining tables and other homogeneous sets of repeated data items. Whenever the OCCURS clause is used, the data-name which is the subject of this entry must be subscripted whenever it is referred to in a statement other than SEARCH. Further, if the subject of this entry is a group item, all data-names belonging to the group must be subscripted whenever they are used as operands, except as the object of a REDEFINES clause.

The OCCURS clause cannot be specified in a data description entry that:

- Has an 01, 66, 77, 78, or 88 level-number.
- Has a variable-occurrence data item subordinate to it.

Except for the OCCURS clause itself, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

The number of occurrences of the subject entry is defined as follows:

- In Format 1, the value of *integer-2* represents the exact number of occurrences.
- In Format 2, the current value of the data item referenced by *data-name-3* represents the number of occurrences.

  This format specifies that the subject of this entry has a variable number of occurrences. The value of *integer-2* represents the maximum number of occurrences and the value of *integer-1* represents the minimum number of occurrences. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable.

  At the time of reference to the subject of this entry or to any containing or subordinate data item, the value of the data item referenced by *data-name-3* must fall within the range *integer-1* through *integer-2*. The contents of the data items whose occurrence numbers exceed the value of the data item referenced by *data-name-3* are undefined.

  When both *integer-1* and *integer-2* are used, the value of *integer-1* must be less than the value of *integer-2*. The value of *integer-1* may be zero. If *integer-1* is omitted, it is assumed to be zero.

  The data description of *data-name-3* must describe an integer. *data-name-3* may be qualified.

  A data description entry that contains Format 2 of the OCCURS clause may be followed, within that record description, only by data description entries that are subordinate to it.

When a group data item having subordinate to it an entry that specifies Format 2 of the OCCURS clause is referenced, the part of the table area used in the operation is determined as follows:

1. If the data item referenced by *data-name-3* is outside the group, only that part of the table area that is specified by the value of the data item referenced by *data-name-3* at the start of the operation is used.

2. If the data item referenced by *data-name-3* is included in the same group and the group data item is referenced as a sending item, only that part of the table area that is specified by the value of the data item referenced by *data-name-3* at the start of the operation is used in the operation. If the group is a receiving item, the maximum length of the group is used.

If Format 2 is specified in a record description entry and the associated file description or sort-merge description entry contains the VARYING phrase of the RECORD clause, the

records are variable length. If the DEPENDING ON phrase of the RECORD clause is not specified, the content of the data item referenced by *data-name-3* of the OCCURS clause must be set to the number of occurrences to be written before the execution of any RELEASE, REWRITE or WRITE statement.

In the KEY IS phrase, the first specification of *data-name-4* must be the name of either the entry containing the OCCURS clause or an entry subordinate to it. Subsequent specifications of *data-name-4* must be subordinate to the entry containing the OCCURS clause. Each *data-name-4* may be qualified, but must not be subscripted, as is normally required. For each *data-name-4*, the associated data description must not include an OCCURS clause, except when the first *data-name-4* is the same as the entry containing the OCCURS clause. There may not be any OCCURS clauses between this OCCURS clause and the descriptions of any *data-name-4*.

When the KEY IS phrase is specified, the repeated data must be arranged in ascending or descending order according to the values contained in *data-name-4*. The ascending or descending order is determined according to the rules for comparison of operands. The data-names are listed in their descending order of significance.

An INDEXED BY phrase may be used to define one or more index-names to be associated with the subject of this entry. Index-names are not data-names, and they may be used only in contexts where the formats explicitly mention them. An index-name is a user-defined word, and each index-name must be unique within the program. Index-names are used principally in subscripts, and their use in this context can result in more efficient access to the elements of a table.

PICTURE Clause

$$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\} \text{IS } \textit{character-string-1}$$

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

A PICTURE clause can be specified only at the elementary item level.

character-string-1 consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item. The maximum number of characters allowed in the character-string is 30.

The lowercase letters corresponding to the uppercase letters representing the PICTURE symbols **A**, **B**, **P**, **S**, **V**, **X**, **Z**, **CR** and **DB** are equivalent to their uppercase representations in a PICTURE character-string. Other lowercase letters are not equivalent to their

corresponding uppercase representations. This means that if a lowercase q, for example, has been designated as the currency symbol the uppercase Q may not be substituted for it, and vice versa.

The PICTURE clause must not be specified for the subject of a RENAMES clause or for an item whose usage is index or pointer. For any other entry describing an elementary item, a PICTURE clause must be specified except that the PICTURE clause may be omitted for an elementary item when an alphanumeric literal is specified in the VALUE clause. In the latter case, a PICTURE clause is implied of the form 'PICTURE X(*length*)', where *length* is the length of the literal specified in the VALUE clause.

PIC and PICTURE are synonymous.

The asterisk when used as the zero suppression symbol and the BLANK WHEN ZERO clause may not appear in the same entry.

Character-String (Data Categories)

The five categories of data that can be described with a PICTURE clause are defined as follows:

1. **Alphabetic.** Its PICTURE character-string can contain only the symbol **A**. The contents of an alphabetic data item when represented in standard data format must be one or more alphabetic characters (“a” through “z”, “A” through “Z”, and space).

2. **Numeric.** Its PICTURE character-string can contain only the symbols **9**, **P**, **S**, and **V**. Its PICTURE character-string must contain at least one symbol **9** and not more than thirty symbols **9**. Each symbol 9 specifies a digit position. If unsigned, the contents of a numeric data item when represented in standard data format must be one or more numeric characters. If signed, a numeric data item may also contain a “+”, “–“, or other representation of an operational sign. The actual in-memory contents of a numeric data item are not standard data format when the usage is other than DISPLAY as specified by a USAGE clause applicable to the data description entry or when the data item is signed and the SEPARATE CHARACTER phrase is not specified in a SIGN clause applicable to the data description entry.

3. **Alphanumeric.** Its PICTURE character-string is restricted to certain combinations of the symbols **A**, **X**, and **9**, and the item is treated as if the character-string contained all symbols **X**. The PICTURE character-string must contain at least one symbol **X** or a combination of the symbols **A** and **9**. A PICTURE character-string that contains all symbols **A** or all symbols **9** does not define an alphanumeric data item, since such character-strings define an alphabetic or numeric data item, respectively. The contents of an alphanumeric data item when represented in standard data format must be two or more characters in the character set of the computer.

4. **Alphanumeric edited.** Its PICTURE character-string is restricted to certain combinations of the following symbols: **A**, **X**, **9**, **B**, **0**, and slash (/). The PICTURE character-string must contain at least one symbol **A** or **X** and at least one symbol **B**, **0**, or slash (/). The contents of an alphanumeric edited date item when represented in standard data format must be two or more characters in the character set of the computer.

5. **Numeric edited.** Its PICTURE character-string is restricted to certain combinations of the following symbols: **B**, slash (/), **P**, **V**, **Z**, **0**, **9**, comma (**,**), period (**.**), asterisk (*****), minus (**–**), plus (**+**), **CR**, **DB**, and the currency symbol (the symbol **$** or the symbol specified in the CURRENCY SIGN clause of the SPECIAL-NAMES paragraph). The allowable combinations are determined from the order of precedence of symbols (see Table 4-4 on page 4-45) and the editing rules, which begin on page 4-40. The number of digit positions that can be represented in the PICTURE character-string must range from one to thirty, inclusive. The character-string must contain at least one symbol **0**, **B**, slash, **Z**, asterisk, plus, minus, comma, period, **CR**, **DB**, or the currency symbol. The contents of each of the character positions in a numeric edited data item must be consistent with the corresponding PICTURE symbol.

Note The additional data categories, **index data** and **data pointer**, also exist, but do not use a PICTURE clause in their data description entry. An index data item is described with the USAGE IS INDEX clause. A data pointer data item is described with the USAGE IS POINTER clause.

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 unsigned nonzero integer which is enclosed in parentheses following the symbol **A**, comma (**,**), **X**, **9**, **P**, **Z**, asterisk (*****), **B**, slash (/), **0**, plus (**+**), minus (**–**), or the currency symbol indicates the number of consecutive occurrences of the symbol. Note that the following symbols may appear only once in a given PICTURE: **S**, **V**, period (**.**), **CR**, and **DB**.

Symbols

The functions of the symbols used in a PICTURE character-string to describe an elementary item are as follows:

A Each symbol **A** in the character-string represents a character position that can contain only an alphabetic character (“a” through “z”, “A” through “Z”, and space). Each symbol **A** is counted in the size of the data item described by the PICTURE character-string.

B Each symbol **B** in the character-string represents a character position into which the character space will be inserted when the data item is the receiving item of an elementary MOVE statement. Each symbol B is counted in the size of the data item described by the PICTURE character-string.

P Each symbol **P** in the character-string 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 symbol **P** is not counted in the size of the data item described by the PICTURE character-string, but each symbol **P** is counted in determining the maximum number (30) of digit positions in numeric or numeric edited data items. The symbol **P** may appear only as a continuous string in the leftmost or rightmost digit positions within a PICTURE character-string. Since the scaling position symbol **P** implies an assumed decimal point (to the left of the symbols **P** if they are the leftmost digit positions and to the right of the symbols **P** if they are the rightmost digit positions), the assumed decimal point symbol **V** is redundant either to the left or right of the symbols **P**, respectively, within such a PICTURE character-string. The symbol **P** and the insertion symbol period (**.**) cannot both occur in the same PICTURE character-string.

In certain operations that reference a data item whose PICTURE character-string contains the symbol **P**, the algebraic value of the data item is used rather than the actual character representation of the data item. This algebraic value assumes the decimal point in the prescribed location and zero in place of the digit positions specified by the symbol **P**. The size of the value is the number of digit positions represented by the PICTURE character-string. These operations are any of the following:

- Any operation requiring a numeric sending operand.
- A MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol **P**.
- A MOVE statement where the sending operand is a numeric edited data item and its PICTURE character-string contains the symbol **P** and the receiving operand is numeric or numeric edited.
- A comparison operation where both operands are numeric.
- In all other operations the digit positions specified with the symbol **P** are ignored and are not counted in the size of the operand.

S The symbol **S** is used in a character-string to indicate the presence, but neither the representation nor, necessarily, the position of an operational sign. The symbol **S** must be written as the leftmost character in the PICTURE character-string. The symbol **S** is not counted in determining the size (in terms of standard data format characters) of the data item described by the PICTURE character-string unless the

entry contains or is subject to a SIGN clause that specifies the SEPARATE CHARACTER phrase. The symbol **S** in the PICTURE character-string and the BLANK WHEN ZERO clause may not occur in the same data description entry.

V The symbol **V** is used in a character-string to indicate the location of the assumed decimal point and may appear only once in any single PICTURE character-string. The symbol **V** does not represent a character position and, therefore, is not counted in the size of the data item described by the PICTURE character-string. When the assumed decimal point is to the right of the rightmost symbol in the string representing a digit position or scaling position, or is to the left of scaling positions that represent the leftmost digit positions, the **V** is redundant. The symbol **V** and the insertion symbol period (**.**) cannot both occur in the same PICTURE character-string.

X Each symbol **X** in the character-string is used to represent a character position that contains any allowable character from the character set of the computer. Each symbol **X** is counted in the size of the data item described by the PICTURE character-string.

Z Each symbol **Z** in a character-string may only be used to represent the leftmost leading numeric character positions that will be replaced by space characters when the contents of those character positions are leading zeroes and the data item is the receiving item of an elementary MOVE statement. Each symbol **Z** is counted in the size of the item described by the PICTURE character-string and in determining the maximum number (30) of digit positions allowed in a numeric edited data item. If the symbol **Z** is used to the right of the decimal point in a character-string, then all digit positions in that character-string must be described with the symbol **Z**. If the symbol **Z** represents all the digit-positions in the character-string, then the described data item is blank when zero, even if the BLANK WHEN ZERO clause is not specified.

9 Each symbol **9** in the character-string represents a character position that contains a numeric character. Each symbol **9** is counted in the size of the item described by the PICTURE character-string and in determining the maximum number (30) of digit positions in a numeric or numeric edited data item.

0 Each symbol **0** in the character-string represents a character position into which the character zero (“0”) will be inserted when the data item is the receiving item of an elementary MOVE statement and removed when a numeric edited data item is the sending item in an elementary MOVE statement with a numeric or numeric edited receiving data item. Each symbol **0** is counted in the size of the data item described by the PICTURE character-string. The symbol **0** does not represent a digit position in a numeric edited data item.

/ Each symbol slash (/) in the character-string represents a character position into which a character slash ("/") will be inserted when the data item is the receiving item of an elementary MOVE statement. Each symbol slash (/) is counted in the size of the data item described by the PICTURE character-string.

, Each symbol comma (**,**) in the character-string represents a character position into which a character comma (",") will be inserted when the data item is the receiving item of an elementary MOVE statement. Each symbol comma (**,**) is counted in the size of the data item described by the PICTURE character-string.

. When the symbol period (**.**) appears in the character-string, it is an editing symbol that represents the decimal point for alignment purposes and, in addition, represents a character position into which the character period (".") will be inserted. The symbol period (**.**) is counted in the size of the data item described by the PICTURE character-string.

Note For a given program the functions of the period and comma are exchanged if the DECIMAL-POINT IS COMMA clause 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 a PICTURE clause.

+, –, CR, DB

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 PICTURE character-string and each character used in the symbol is counted in determining the size of the data item described by the PICTURE character-string. If the symbols plus or minus occur more than once (a floating sign control symbol), then one less than the total number of these symbols is counted in determining the maximum number (30) of digit positions allowed in a numeric edited data item. If a floating symbol plus or minus is used to the right of the decimal point in a character-string, then all digit positions in that character-string must be described with the symbol plus or minus, respectively. If a floating plus or minus symbol string represents all the digit-positions in the character-string, then the described data item is blank when zero, even if the BLANK WHEN ZERO clause is not specified.

***** Each symbol asterisk (*****) in the character-string represents a leading numeric character position into which a character asterisk ("*") will be placed when that position contains a leading zero and the data item is the receiving item of an elementary MOVE statement. Each symbol asterisk (*****) is counted in the size of the data item described by the PICTURE character-string and in determining the maximum number (30) of digit positions allowed in a numeric edited data item. If the symbol asterisk (*****) is used to the right of the decimal point in a character-string, then all digit positions in that character-string must be described with the symbol asterisk (*****). The symbol asterisk in the PICTURE character-string and the BLANK

WHEN ZERO clause may not occur in the same data description entry. If the symbol asterisk represents all the digit-positions in the character-string, then, when zero, the described data item is all asterisks (ALL "*"), except that, if the character-string contains the symbol period (**.**), a character period (".") will occur at the specified location in the data item.

cs The currency symbol in a character-string is represented either by the currency sign (the symbol **$**) or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol in the character-string represents a character position into which a currency symbol is to be placed when the data item is the receiving item of an elementary MOVE statement. Each currency symbol is counted in the size of the data item described by the PICTURE character-string. If the currency symbol occurs more than once (a floating currency symbol), then one less than the total number of currency symbols is counted in determining the maximum number (30) of digit positions allowed in a numeric edited data item. If the currency symbol is used to the right of the decimal point in a character-string, then all digit positions in that character-string must be described with the currency symbol. If a floating currency symbol string represents all the digit-positions in the character-string, then the described data item is blank when zero, even if the BLANK WHEN ZERO clause is not specified.

Editing Rules

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

1. Simple insertion
2. Special insertion
3. Fixed insertion
4. Floating insertion

There are two types of suppression and replacement editing:

1. Zero suppression and replacement with spaces
2. Zero suppression and replacement with asterisks

The type of editing which may be performed upon a data item depends on the category to which the data item belongs. Table 4-1 specifies which type of editing may be performed upon a given category.

Table 4-1: PICTURE Clause Editing

Category	Type of Editing
Alphabetic	None.
Numeric	None.
Alphanumeric	None.
Alphanumeric Edited	Simple insertion using symbols 0, B, and slash (/).
Numeric Edited	All, subject to the following rules.

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

The symbols comma (**,**), **B**, **0**, and slash (/) 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.

Special Insertion Editing

The symbol period (**.**) is used as the insertion character. It 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 symbol period (**.**), in the same PICTURE character-string is disallowed. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

Fixed Insertion Editing

The currency symbol and the editing sign control symbols plus (+), minus (–), **CR**, and **DB** are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols **CR** and **DB** are used, they represent two character positions in determining the size of the item, and they must represent the rightmost character positions that are counted in the size of the item. If these character positions contain the symbols **CR** or **DB**, the uppercase letters are the insertion characters.

A plus (+) or minus (–) symbol, when used, must be either the leftmost or rightmost character position to be counted in the size of the item.

The currency symbol must be the leftmost character position to be counted in the size of the item except that it can be preceded by either a plus (+) or a minus (–) symbol.

Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character-string.

Editing sign control symbols produce the results shown in Table 4-2 depending upon the value of the data item.

Table 4-2: Editing Symbol Results

Editing Symbol	Result	
	Data Item (Positive or Zero)	Data Item (Negative)
+	+	–
–	space	–
CR	2 spaces	CR
DB	2 spaces	DB

Floating Insertion Editing

The currency symbol and editing sign control symbols plus (+) and minus (–) 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. The string may contain any of the simple insertion symbols or have simple insertion characters immediately to its right. Such simple insertion characters are part of the floating string. When the floating insertion character is the currency symbol, the string of floating insertion characters may have the fixed insertion characters CR and DB immediately to the right of the string.

The leftmost character of the floating insertion string represents the leftmost limit of the floating symbols 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. One way is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other way is to

represent all of the numeric character positions in the PICTURE character-string by the insertion character.

If the insertion characters are only to the left of the decimal point in the PICTURE character-string, the result is that a single floating insertion character will be placed into 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, at least one of the insertion characters must be to the left of the decimal point.

When the floating character is the editing control symbol plus (+) or minus (–), the character inserted depends upon the value of the data item; see Table 4-3.

Table 4-3: Results of + and – Editing

	Result	
Editing Symbol	**Data Item (Positive or Zero)**	**Data Item (Negative)**
+	+	–
–	space	–

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon 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.

Zero Suppression Editing

The suppression of leading zeroes in numeric character positions is indicated by the use of the symbol **Z** or by the symbol 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; if the asterisk is used, the replacement character will be *.

Zero suppression and replacement are indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the associated character position in the data contains a leading 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 which 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, including any editing characters, is spaces. If the value is zero and the suppression symbol is asterisk (*****), the entire data item, including any insertion editing symbols except the actual decimal point, is "*". In this case, the actual decimal point will appear in the data item.

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

PICTURE Symbol Precedence

Table 4-4 shows the order of precedence when using characters as symbols in a PICTURE character-string. An "X" at an intersection indicates that the symbol (or symbols) at the top of the column may precede (but not necessarily immediately), in a given character-string, the symbol (or symbols) at the left of the row. Arguments listed as one or another (for instance, plus (+) or minus (–)) indicate mutually exclusive symbols. The currency symbol is indicated by the symbol **cs**.

At least one of the symbols **A**, **X**, **Z**, **9** or asterisk (*****), or at least two occurrences of one of the symbols plus (+), minus (–), or **cs** must be present in a PICTURE character-string.

The nonfloating insertion symbols plus (+) and minus (–), the floating insertion symbols **Z**, asterisk (*****), plus (+), minus (–), and **cs**, and the symbol **P** appear twice in Table 4-4. The first appearance of the symbol in the FIRST SYMBOL column and SECOND

SYMBOL row represents its use to the left of the decimal point position. The second appearance of the symbol represents its use to the right of the decimal point position.

Table 4-4: PICTURE Symbol Precedence

Second Symbol \ 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
Non-floating Insertion Symbols	B	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
	{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		
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

REDEFINES Clause

level-number-1 [*data-name-1* | FILLER]

[REDEFINES *data-name-2*]

The REDEFINES clause allows a computer storage area to be described by different data description entries.

Note *level-number-1*, *data-name-1* and FILLER are shown in the above format (gray highlight) to improve clarity. They are not part of the REDEFINES clause.

The level-numbers of *data-name-1* and *data-name-2* must be identical but must not be 66 or 88.

This clause must not be used in level 01 entries in the File Section or Communication Section.

The data description entry for *data-name-2* cannot contain an OCCURS clause. However, *data-name-2* may be subordinate to an item whose data description contains an OCCURS clause. In this case, the reference to *data-name-2* in the REDEFINES clause may not be subscripted. Neither the original definition nor the redefinition can include a variable-occurrence data item.

data-name-2 must not be qualified; if it is not a unique data-name, the necessary qualification is implicitly provided by the position of the REDEFINES clause within the hierarchical structure of the Data Division.

No entry having a level-number numerically lower than the level-number of *data-name-2* and the subject of the entry may occur between the data description entries of *data-name-2* and the subject of the entry.

Redefinition starts at *data-name-2* and ends when a level-number less than or equal to that of *data-name-2* is encountered.

When the level-number of *data-name-1* is other than 01, it must not specify more character positions than the data item referenced by *data-name-2* contains. It is important to observe that the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

Multiple redefinitions of the same character positions are permitted. When multiple redefinitions are used, either the first or the most recently defined name on the same level within the current hierarchy may be used as *data-name-2*.

The entries giving the new description of the character positions must not contain any VALUE clauses except in condition-name entries.

Multiple level 01 entries subordinate to any given level indicator represent implicit redefinitions of the same area.

RENAMES Clause

66 *data-name-1*

RENAMES *data-name-2* [{ THROUGH / THRU } *data-name-3*] .

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

Note Level-number 66 and *data-name-1*, and the period space separator are shown in the above format (gray highlight) to improve clarity. They are not part of the RENAMES clause.

All RENAMES entries referring to data items within a given logical record must immediately follow the last data description entry of the associated record description entry.

data-name-2 and *data-name-3* must be names of elementary items or groups of elementary items in the same logical record, and 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 level entry.

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 entry. Neither *data-name-2* nor *data-name-3* may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry.

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*.

data-name-2 and *data-name-3* may be qualified.

None of the items within the range, including *data-name-2* and *data-name-3*, if specified, can be variable-occurrence data items.

One or more RENAMES entries can be written for a logical record.

When *data-name-3* is not specified, all of the data attributes of *data-name-2* become the data attributes for *data-name-1*, and *data-name-1* may be used as a synonym for *data-name-2*.

When *data-name-3* is specified, *data-name-1* is defined as 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), and concluding with *data-name-3* (if *data-name-3* is an elementary item) or the last elementary item in *data-name-3* (if *data-name-3* is a group item).

The words THRU and THROUGH are synonymous.

SIGN Clause

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

The SIGN clause may be used to specify the position and mode of representation of the operational sign for signed numeric data items. It may be specified either at the elementary level or at the group level. When it is specified at the elementary level, it applies only to that item. When it is specified at the group level, it applies to each subordinate signed numeric data item.

If a SIGN clause is specified in a group item subordinate to another group item that also has a SIGN clause, the SIGN clause specified in the subordinate group item takes precedence for that subordinate group item.

If a SIGN clause is specified in an elementary numeric data description entry subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate elementary numeric data description entry takes precedence for that elementary numeric data item.

The SIGN clause is applicable only to numeric data description entries whose PICTURE character-string contains the symbol S and whose explicit or implicit usage is DISPLAY.

If the CODE-SET clause is specified, any signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause.

A numeric data description entry whose PICTURE character-string contains the character S, but to which no optional SIGN clause applies, has an operational sign

whose representation depends on the setting of the S (Separate Sign) Compile Command Option:

- If the Separate Sign Default option is not in effect, the operational sign will be the same as if SIGN IS TRAILING (without the optional SEPARATE CHARACTER phrase) had been specified.
- If the Separate Sign Default option is in effect, the operational sign will be the same as if SIGN IS TRAILING SEPARATE CHARACTER had been specified.

See Chapter 6, *Compiling*, of the *RM/COBOL User's Guide*, for a full discussion of the S Compile Command Option.

If the optional SEPARATE CHARACTER phrase is not present:

- The operational sign will be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item.
- 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).
- The valid signs for combined sign data items depend on the value of the leading (or, respectively, trailing) digit with which the sign is associated and whether the value is positive or negative; see Table 4-5.

Table 4-5: Valid Data Item Encodings

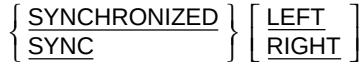
Digit	Positive Value Valid Encodings	Negative Value Valid Encodings
0	{	}
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R

See the USAGE clause on page 4-52 for the valid sign values on other data types.

If the optional SEPARATE CHARACTER phrase is present, then:

- The operational sign will be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.
- The letter S in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).
- The operational signs for positive and negative are the standard data format characters + and –, respectively.

SYNCHRONIZED Clause



The SYNCHRONIZED clause specifies the alignment of an elementary item on an even byte boundary.

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 delimiting 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) are not used for any other data item. Such unused character positions, however, are included in:

- The size of any group item (or items) to which the elementary item belongs.
- The character positions redefined when this data item is the object of a REDEFINES clause.

The words SYNC and SYNCHRONIZED are synonymous.

This clause may appear only with an elementary item or with the USAGE IS INDEX clause.

SYNCHRONIZED LEFT specifies that the elementary item is to be positioned such that it will begin at the next available even byte boundary. If the data item contains an odd number of character positions, one trailing character position of FILLER is supplied.

SYNCHRONIZED not followed by either RIGHT or LEFT is equivalent to a SYNCHRONIZED LEFT clause.

SYNCHRONIZED RIGHT specifies that the elementary item is to be positioned such that it will terminate on an even byte boundary. If the data item contains an odd number of character positions, a leading character position of FILLER is supplied.

Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as determined by the PICTURE clause, the USAGE clause and the SIGN clause, is used in determining any action that depends on size, such as justification, truncation or overflow.

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, without respect to whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

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:

- Each occurrence of the data item is SYNCHRONIZED.
- Any implicit FILLER generated for other data items within that same table are generated for each occurrence of those data items.

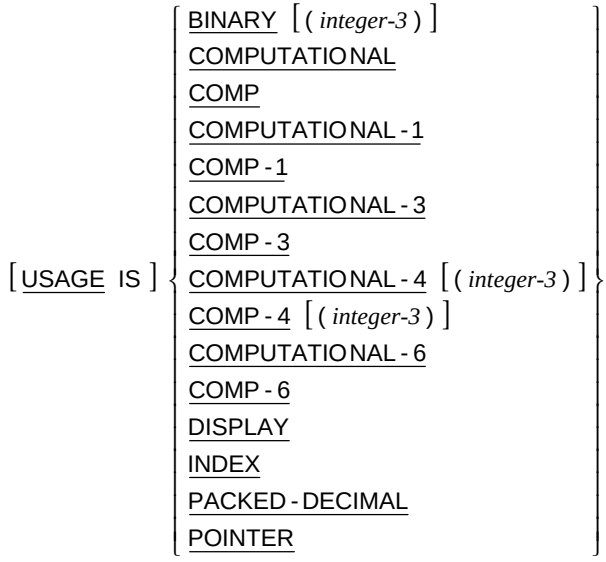
Records of a file and index data items are automatically synchronized left. Records and noncontiguous data items in working storage are implicitly synchronized left unless explicitly synchronized right.

The format on external media of records or groups containing elementary items described with the SYNCHRONIZED clause includes any implied FILLER bytes.

When the data item preceding a data item described with the SYNCHRONIZED clause does not terminate on a byte whose address is even, one implied character position of FILLER is generated. Such automatically generated FILLER positions are included in:

- The size of any group to which the FILLER item belongs.
- The number of character positions allocated when the group item of which the FILLER item is a part appears as the object of a REDEFINES clause.

USAGE Clause



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

integer-3 must be in the range 1 through 16 and represents the number of bytes to allocate as a binary allocation override. The binary allocation override may also be specified following COMPUTATIONAL or COMP usage if the compiler has been configured to treat this usage type as binary by use of the COMPUTATIONAL-TYPE keyword of the COMPILER-OPTIONS configuration record.

This clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referenced.

The USAGE clause may 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. When the usage is COMPUTATIONAL-4, COMP-4, or BINARY, the binary allocation override may also be included or omitted at any level as part of a USAGE clause at that level. If included, the binary allocation override may specify a different value than that specified in the USAGE clause for a containing group. The rules for the binary allocation override within group structure are as follows:

- If the USAGE clause is written at a group level and specifies COMPUTATIONAL-4, COMP-4, or BINARY usage with the allocation override *integer-3* specified in

parentheses following the usage type, the allocation override *integer-3* applies to each elementary item in the group that does not specify a USAGE clause and is not subordinate to another group with a higher level-number that specifies a USAGE clause.

- If the USAGE clause is written at a group level and specifies COMPUTATIONAL-4, COMP-4, or BINARY usage without the allocation override *integer-3*, the configured binary allocation scheme applies to each elementary item in the group that does not specify a USAGE clause and is not subordinate to another group with a higher level-number that specifies a USAGE clause.

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.

A COMPUTATIONAL (COMPUTATIONAL-1, COMPUTATIONAL-3, COMPUTATIONAL-4, COMPUTATIONAL-6) item represents a value to be used in computations and must be numeric. If a group is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. The group itself is not COMPUTATIONAL (cannot be used in computations).

The PICTURE character-string of a COMPUTATIONAL, BINARY, COMPUTATIONAL-3, COMPUTATIONAL-4, or PACKED-DECIMAL item can contain only 9's, the operational sign character S, the implied decimal point character V and one or more P's. A COMPUTATIONAL-1 item must be an integer. Therefore, its PICTURE character-string may not contain any P's; if V is used it must be the rightmost character in the PICTURE character-string. The PICTURE character-string of a COMPUTATIONAL-6 item can contain only V, 9, and P.

COMPUTATIONAL Usage

The format of a COMPUTATIONAL (abbreviated COMP) item is one decimal digit per character position for each 9 in the PICTURE character-string. If an S appears in the PICTURE character-string, an additional trailing character contains the sign. Hexadecimal values 0 through 9 are used to represent the numeric digits zero through nine. The hexadecimal value D is used for negative sign representation. Depending on configured sign representation, the hexadecimal value C or F is used for positive sign representation. Any value other than the correct values for numeric digit or sign representation will be treated as invalid for purposes of the NUMERIC class condition.

Note The compiler may be configured to treat COMPUTATIONAL (COMP) usage as if it were BINARY or PACKED-DECIMAL usage by use of the COMPUTATIONAL-TYPE keyword of the COMPILER-OPTIONS configuration record.

COMPUTATIONAL-1 Usage

The format of a COMPUTATIONAL-1 item (abbreviated COMP-1) is 16-bit two's complement signed binary integer, independent of the number of 9's or the appearance of S in the PICTURE character-string. The number of 9's is significant when the value is converted to decimal during data manipulation and for the size error condition. The value of a COMPUTATIONAL-1 item ranges between –32768 and 32767.

When a COMP-1 data item is specified as a receiving operand in an arithmetic statement, the size error condition can be caused by either of the following:

- The PICTURE character-string specifies fewer than five 9's, and the decimal representation of the value to be stored into the COMP-1 data item requires more 9's than are specified.
- The PICTURE character-string specifies five or more 9's, and the value to be stored into the COMP-1 data item falls outside the range –32768 through 32767, inclusive.

COMPUTATIONAL-3 or PACKED-DECIMAL Usage

The format of a COMPUTATIONAL-3 (abbreviated COMP-3) and PACKED-DECIMAL item is two decimal digits per character position. All COMPUTATIONAL-3 and PACKED-DECIMAL items are allocated a field for an operational sign whether or not an S appears in the PICTURE character-string. The operational sign field occupies the rightmost four bits (half of a character position) of the item. The digits of the data item occupy half-character positions immediately to the left of the operational sign field, one for each 9 in the PICTURE character-string. If an even number of 9's are in the PICTURE character-string of the data item, an additional half-character is allocated at the left end of the item, to complete an integral number of character positions. This extra position is not available for storage of a digit when the data item is used as a receiving field. Nor is it used when determining the size error condition or when validating VALUE IS literals.

If the PICTURE character-string of a COMPUTATIONAL-3 or PACKED-DECIMAL item contains an S, the value of the operational sign field is used to indicate the sign of the data item. Hexadecimal values 0 through 9 are used to represent numeric digits. The hexadecimal value D is used for negative sign representation. Depending on configured sign representation, the hexadecimal value C, B or F is used for positive sign representation. Any value other than the correct values for numeric digit or sign representation will be treated as invalid for purposes of the NUMERIC class condition.

If the PICTURE character-string of a COMPUTATIONAL-3 or PACKED-DECIMAL item does not contain an S, the data item is treated as nonnegative regardless of the contents of the operational sign field.

COMPUTATIONAL-4 or BINARY Usage

The format of a COMPUTATIONAL-4 (abbreviated COMP-4) or BINARY item is binary with the high order bytes at lower addresses than the low-order bytes. Twos-complement binary is used to represent signed data items, that is, when the PICTURE character-string of a COMPUTATIONAL-4 or BINARY item contains an S. If an allocation override *integer-3* is specified in parentheses following the usage type, then *integer-3* bytes will be allocated. Otherwise, the number of bytes allocated depends on the number of 9's in the PICTURE character-string and the BINARY-ALLOCATION and BINARY-ALLOCATION-SIGNED keywords of the COMPILER-OPTIONS configuration record. The default configured binary allocation scheme is two bytes for one to four 9's, four bytes for five to nine 9's, eight bytes for ten to eighteen 9's, and sixteen bytes for nineteen to thirty 9's.

The value of *integer-3* in a binary allocation override may be less than the number of bytes required to support the decimal precision specified in the PICTURE character-string by the number of 9's included in that character-string. In this case, the size error condition will be detected if the value to be stored is outside the range of values supported by the number of bytes indicated by *integer-3*. For example, an item described as PIC S9(3) BINARY (1) can store values in the range -128 to +127; a size error condition would exist on an attempt to store values less than -128 or greater than +127 into such an item. As another example, an item described as PIC 99V9 BINARY (1) can store values in the range 0 to 25.5; a size error condition would exist on an attempt to store a value less than -25.5 or greater than +25.5 into such an item. Note that in this latter example, when a negative value is the sending value, its absolute value is stored into the data item because the item is unsigned.

The binary allocation override does not increase the precision specified by the PICTURE character-string. For example, the specification PIC 9 BINARY(1) describes a data item that will cause the size error condition (in arithmetic statements) or truncation (in MOVE statements) for numbers greater than 9, except in those cases where truncation does not apply (as in a group move or an arithmetic statement that does not specify the SIZE ERROR phrase). Thus, while it is possible to have numbers with values from 0 to 255 in this data item, the programmer should only plan on being able to put values from 0 to 9 in this data item.

COMPUTATIONAL-6 Usage

The COMPUTATIONAL-6 type (abbreviated COMP-6) is used for describing unsigned packed decimal internal representation of numeric data. The format of a COMPUTATIONAL-6 data item is two decimal digits per character position. All COMPUTATIONAL-6 items are unsigned and must not contain an S in the PICTURE character-string; the format does not reserve any space for an operational sign. If an odd number of 9's are in the PICTURE character-string of the data item, an additional half-character is allocated at the left end of the item, to complete an integral number of character positions. This extra position is not available for storage of a digit when the data item is used as a receiving field. Nor is it used when determining the size error condition or when validating VALUE IS literals.

DISPLAY Usage

The USAGE IS DISPLAY clause indicates that the format of the data is the standard data format and that the data is aligned on a character boundary. The operational sign of a signed DISPLAY item is determined by whether the data item has a separate or combined sign. If the data item has a separate sign, a + in the sign field indicates a positive value and a – indicates a negative value. Any other value is considered invalid for purposes of the NUMERIC class condition. If the data item has a combined sign, refer to Table 4-5 on page 4-49 for valid sign encoding.

INDEX Usage

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. If a group item is described with the USAGE IS INDEX clause the elementary items in the group are all index data but the group item name cannot be used in the SET statement or in a relation condition.

An index data item can be referenced explicitly only in a SEARCH or SET statement, a relation condition, the USING phrase of a Procedure Division header, or the USING phrase of a CALL statement. An index data item may not be a conditional variable.

Index data items are implicitly synchronized left.

The JUSTIFIED, PICTURE, VALUE and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

An index data item can be part of a group which is referred to in a MOVE statement or in an Input-Output statement, in which case no conversion will take place.

POINTER Usage

The USAGE IS POINTER clause indicates that the data item is to contain a pointer to another data item. A pointer is a 24-byte structure that includes address, offset, and length fields. This structure is implementation-dependent, and the individual fields are not meant to be manipulated except with Formats 5 and 6 of the SET statement. POINTER usage data items have no PICTURE clause and thus are not numeric operands.

The effective address of a pointer is the sum of its address and offset fields. A pointer covers an area of memory from the value of the address field to the sum of the values of the address and length fields.

Pointer data items should never be described in record description entries in the File Section since their value is only valid during a particular run unit.

Pointer data items should not be the subject or object of a redefinition and should only be manipulated with the INITIALIZE statement, Formats 5 and 6 of the SET statement, or set to a valid value by use of the C$MemoryAllocate subprogram in the supplied subprogram library. If a group containing pointer data items is moved to another group, the receiving group should have pointer data items in the same locations within the group as in the sending group. Failure to follow these rules will result in a severely misbehaving program that may terminate the run unit with a memory access violation exception, may change the code memory for the COBOL program or the runtime system, or may inadvertently change data values other than those intended to be changed.

The JUSTIFIED, PICTURE, and BLANK WHEN ZERO clauses may not be used to describe group or elementary items described with the USAGE IS POINTER clause.

The VALUE clause may be used with the USAGE IS POINTER clause, but the only literal that may be specified in such a VALUE clause is the figurative constant NULL (NULLS).

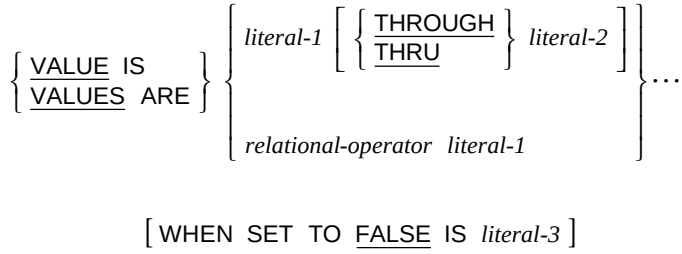
Pointer data items may only be used in relation conditions involving another pointer data item, the USING and GIVING phrases of the Procedure Division header, the INITIALIZE statement, the CALL statement, and Formats 5 and 6 of the SET statement.

VALUE Clause

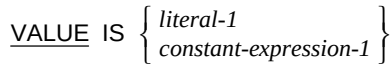
Format 1: Data Item Initialization

```
VALUE IS literal-1
```

Format 2: Condition-Name Values



Format 3: Constant-Name Value



The VALUE clause defines the initial values of Working-Storage Section and Communication Section data items, the values used by the VALUE phrase of the INITIALIZE statement, the values associated with condition-names, the values of Screen Section displayable items, and the values assigned to constant-names.

A signed numeric literal must have associated with it a signed numeric PICTURE character-string.

All numeric literals 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. Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

The words THRU and THROUGH are synonymous.

The WHEN SET TO FALSE phrase has meaning only if the associated condition-name is referenced in a SET *condition-name-1* TO FALSE statement. This phrase declares the false value to be used in a SET *condition-name-1* TO FALSE statement.

When the VALUE clause specifies a nonnumeric literal for an elementary data item, the PICTURE clause may be omitted. In this case, a PICTURE clause of the form 'PICTURE X(*length*)' is implied, where *length* is the length of the nonnumeric literal.

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. The following rules apply:

1. If the category of the item is 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.

2. If the category of the item is alphabetic, alphanumeric, alphanumeric edited or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. 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. Therefore, the value of an edited item is presented in an edited form.

3. As an exception to the rules in item 2, RM/COBOL allows specification of a numeric literal in the VALUE clause for an item of category numeric edited. In this case, the compiler performs the logical equivalent of a MOVE of the numeric literal to the numeric edited data item to form a nonnumeric literal that is used to initialize the numeric edited data item (Format 1) or as the relation literal value for a condition-name (Format 2). If the numeric literal value is zero and a BLANK WHEN ZERO clause applies to the data item, then the resultant nonnumeric literal will be space filled.

4. If the category of the item is data pointer, all literals in the VALUE clause must be pointers. The only pointer literal is NULL (NULLS).

Initialization takes place independent of any BLANK WHEN ZERO or JUSTIFIED clause that may be specified, except as noted in item 3 above.

A figurative constant may be substituted in both Format 1 and Format 2 wherever a literal is specified.

Data Item Initialization Rules (Format 1 VALUE Clause)

Rules governing the use of the VALUE clause differ with the respective sections of the Data Division:

1. In the File Section, the VALUE clause takes effect only during the execution of an INITIALIZE statement. The initial values of the data items in the File Section are undefined.

2. In the Working-Storage Section and Communication Section, the VALUE clause specifies the initial value of data items, other than those data items in an external record. In this case, the VALUE clause causes the data item to assume the specified value when the program is placed into its initial state. For all data items, including data items in an external record, the VALUE clause also takes effect during the execution of an INITIALIZE statement. If the VALUE clause is not used in the description of a data item, the initial value is undefined.

3. In the Linkage Section, the VALUE clause takes effect only during the execution of an INITIALIZE statement. The initial values of the data items in the Linkage Section are specified in the rules for the Linkage Section (see page 4-19) and the Procedure Division (see page 5-1).

The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

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. The VALUE clause must not be written for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY).

If a Format 1 VALUE clause is specified in a data description entry that contains an OCCURS clause or in an entry that is subordinate to an OCCURS clause, each occurrence of the data item is initialized to the specified value.

A data item is said to be associated with a variable-occurrence data item in any of the following circumstances:

- It is a group data item that contains a variable-occurrence data item.
- It is a variable-occurrence data item.
- It is a data item that is subordinate to a variable-occurrence data item.

If a VALUE clause is specified in the data description entry of a data item that is associated with a variable-occurrence data item, the initialization of the data item behaves as if the value of the data item referenced by the DEPENDING ON phrase in the OCCURS clause specified for the variable-occurrence data item had been set to the maximum number of occurrences as specified by that OCCURS clause.

If a VALUE clause is associated with the data item referenced by a DEPENDING ON phrase, that value is considered to be placed into the data item after the variable-occurrence data item is initialized.

Condition-Name Rules (Format 2 VALUE Clause)

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.

Format 2 can be used only in connection with condition-names. Wherever the THROUGH phrase is used, *literal-1* must be less than *literal-2*.

When a relational operator is specified, the truth-value of the condition-name is determined by a relation condition that uses the conditional variable as the subject, *relational-operator* as the relational operator, and *literal-1* as the object of the relation. If a relational operator is specified with the first, or only, *literal-1*, a true value for the purposes of the SET … TO TRUE statement is not defined unless that relational operator includes equality, in which case, *literal-1* is the true value for the SET statement. The relational operators GREATER THAN, >, LESS THAN, <, NOT EQUAL, NOT =, LIKE, and NOT LIKE do not include equality; if the SET … TO TRUE statement is to be used with a condition-name defined with one of these relational operators, the Format 2 VALUE clause must first specify a *literal-1* without a relational operator or with a relational operator that includes equality.

Note The *relational-operator* may be any of the relational operators for the relation condition, which is discussed beginning on page 5-24.

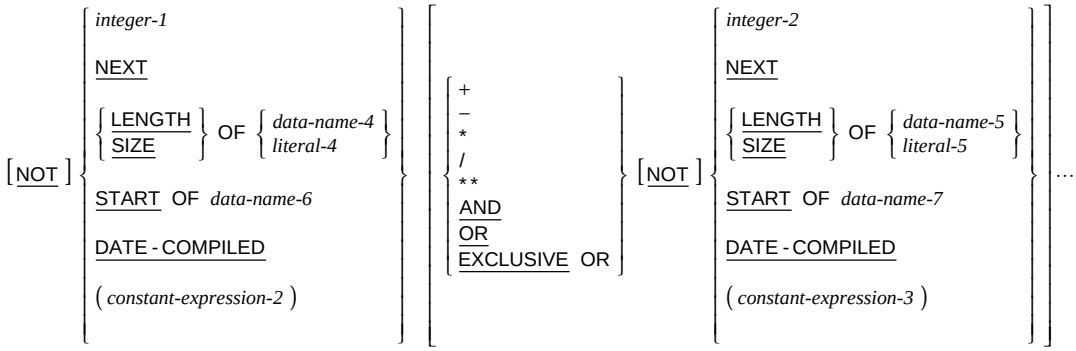
Constant-Name Rules (Format 3 VALUE Clause)

A Format 3 VALUE clause may be used only in a level-number 78 constant-name data description entry.

If *literal-1* is specified, the constant-name has the same characteristics as *literal-1*.

If *constant-expression-1* is specified, the constant-name has the characteristics of an unsigned integer literal.

A constant-expression has the following format:



literal-1 or *integer-1* may be specified by a constant-name previously declared.

integer-1 may be signed or zero, but must be nonnegative and less than 4,294,967,296.

data-name-4, *data-name-5*, *data-name-6*, and *data-name-7* may be qualified.

data-name-4, *data-name-5*, *data-name-6*, and *data-name-7* must have been defined before the declaration of the level-number 78 constant-name. Further, if *data-name-4* or *data-name-5* refer to a group data item, that group must have been completed before the declaration of the level-number 78 constant-name. The group is completed by the specification of another data item at the same or lower level-number than the level-number of the group referenced by *data-name-4*, or *data-name-5*, respectively.

In *constant-expression-1*, any number of arithmetic or logical operators may be used. The result is evaluated using 32-bit integer arithmetic in strict left to right order with all operators having the same precedence. Parentheses may be used to force a desired precedence since constant-expressions in parentheses are evaluated first. If any intermediate result is less than zero, the final value is undefined. Overflow is ignored when evaluating arithmetic operations. The value assigned to a constant-name is included in the compiler listing allocation map, which may be used to verify the results of a constant-expression evaluation as well as simple literal assignments to constant-names.

The logical operators AND, OR, EXCLUSIVE OR, and NOT operate on the binary representation in a bit-wise manner. The binary representation is a 32-bit integer quantity.

LENGTH OF *data-name-4* or SIZE OF *data-name-4* gives the integer value representing the number of character positions allocated for the data item referenced by *data-name-4*. If *data-name-4* is a group item, the length includes all subordinate data items and any filler generated because of SYNCHRONIZED clauses.

LENGTH OF *literal-4* or SIZE OF *literal-4* gives the integer value representing the number of character positions required for *literal-4*. If the literal is a numeric literal, the number of characters is the same as the number of digits. That is, for a numeric literal, the sign and decimal point characters, if specified, are not counted in the length of the literal.

NEXT gives the integer value representing the offset at which the next byte of storage that follows the previous data declaration. If that data description specifies the OCCURS clause and describes an elementary data item, the value given by NEXT is the offset following the maximum number of occurrences specified by the OCCURS clause. If NEXT is used in a level-number 78 entry that is embedded in a group or at the end of a group and that group data description specifies the OCCURS clause, the value given by NEXT is the offset following a single occurrence of the group up to the point the level-number 78 entry is specified.

START OF *data-name-6* gives the integer value representing the offset at which the data item referenced by *data-name-6* begins.

Offsets given by NEXT and START OF are defined as follows:

- If the data item is part of an EXTERNAL record or a LINKAGE record, the offset is calculated from the start of the associated 01-level data item.
- If the data item is part of a FILE record, the offset is calculated from the start of the associated 01-level data item for NEXT or for a START OF specified before the end of the File Section. For a START OF specified after the end of the File Section, the offset is calculated from the beginning of the Data Division. (This difference is caused by the need to handle SAME clauses specified in the I-O-CONTROL paragraph at the end of the File Section.)
- Otherwise, the offset is calculated from the start of the Data Division.

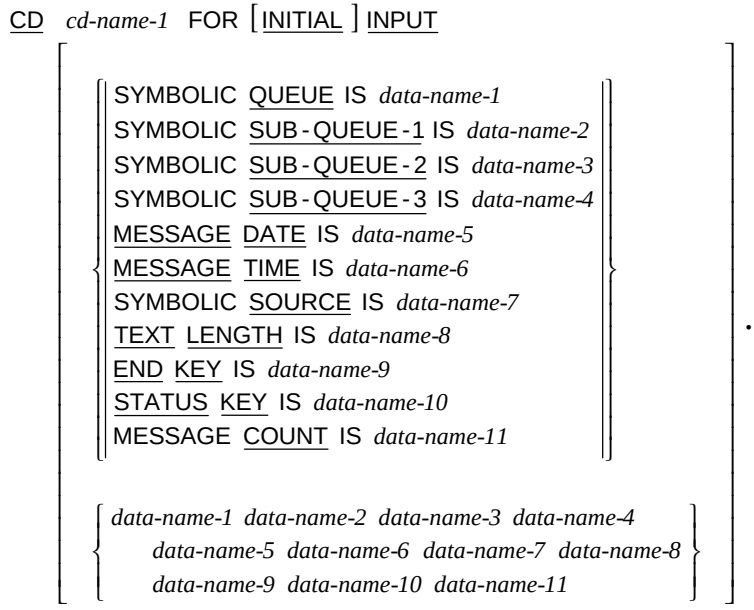
Offsets are not portable across different COBOL implementations, and no reliance should be placed on particular values outside this compilation unit.

DATE-COMPILED gives the integer value representing the date the compilation started as YYYYMMDD, where YYYY is the year, MM is the month of the year, and DD is the day of the month.

Communication Description Entry

The communication description entry specifies the interface area between the Message Control System (MCS) and a COBOL program.

Format 1: Input CD



Format 2: Output CD

```
CD  cd-name-1  FOR OUTPUT
    [DESTINATION COUNT IS data-name-1 ]
    [TEXT LENGTH IS data-name-2 ]
    [STATUS KEY IS data-name-3 ]
    [ DESTINATION TABLE OCCURS integer-1 TIMES
          [INDEXED BY {index-name-1}... ] ]
    [ERROR KEY IS data-name-4 ]
    [SYMBOLIC DESTINATION IS data-name-5 ].
```

Format 3: Input-Output CD

```
CD  cd-name-1  FOR [INITIAL ] I-O
    [
      { MESSAGE DATE IS data-name-1
        MESSAGE TIME IS data-name-2
        SYMBOLIC TERMINAL IS data-name-3
        TEXT LENGTH IS data-name-4
        END KEY IS data-name-5
        STATUS KEY IS data-name-6 }

      { data-name-1 data-name-2 data-name-3 data-name-4
          data-name-5 data-name-6 }
    ] .
```

A CD entry may appear only in the Communication Section.

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.

For Format 1 (Input CD):

- If the INITIAL clause is present it must appear in the position shown; the other optional clauses may be specified in any order.
- If neither option for specifying the interface area is used, a level 01 data description entry must follow the CD entry. Either option may be followed by a level 01 data description entry.
- Record description entries following an input CD entry implicitly redefine the record area established by the input CD entry and must describe a record of exactly 87

standard data format characters. Multiple redefinitions of this record are permitted. VALUE clauses for data items not in the first redefinition do not cause those data items to have an initial value when the program is placed into its initial state, but will be used for the INITIALIZE statement with the VALUE phrase. The MCS always references the record according to the data description defined in item 2k of the general rules for Format 1, which begin on page 4-67.

- *data-name-1*, *data-name-2*, *data-name-3*, *data-name-4*, *data-name-5*, *data-name-6*, *data-name-7*, *data-name-8*, *data-name-9*, *data-name-10* and *data-name-11* must be unique within the CD entry. Within this series any data-name may be replaced by the reserved word FILLER.

For Format 2 (Output CD):

- If none of the optional clauses of the CD entry is specified, a level 01 data description entry must follow the CD entry.

- Record description entries subordinate to an output CD entry implicitly redefine the record area established by the output CD entry. Multiple redefinitions of this record are permitted. VALUE clauses for data items not in the first redefinition do not cause those data items to have an initial value when the program is placed into its initial state, but will be used for the INITIALIZE statement with the VALUE phrase. The MCS always references the record according to the data description defined in item 2d of the general rules for Format 2, which begin on page 4-71.

- *data-name-1*, *data-name-2*, *data-name-3*, *data-name-4* and *data-name-5* must be unique within a CD entry.

- If both the DESTINATION TABLE OCCURS clause and the ERROR KEY clause are present, the ERROR KEY clause must not precede the DESTINATION TABLE OCCURS clause. If both the DESTINATION TABLE OCCURS clause and the SYMBOLIC DESTINATION clause are present, the SYMBOLIC DESTINATION clause must not precede the DESTINATION TABLE OCCURS clause. Except for these restrictions the optional clauses of an output CD entry may appear in any order.

- If the DESTINATION TABLE OCCURS clause is not specified, one error key and one symbolic destination area are assumed. In this case, subscripting is not permitted when referencing these data items.

- If the DESTINATION TABLE OCCURS clause is specified, *data-name-4* and *data-name-5* may be referenced only by subscripting.

For Format 3 (Input-Output CD):

- If the INITIAL clause is present it must appear in the position shown; the other optional clauses may be specified in any order.
- If neither option for specifying the interface area is used, a level 01 data description entry must follow the CD entry. Either option may be followed by a level 01 data description entry.
- Record description entries following an input-output CD entry implicitly redefine the record area established by the input-output CD entry and must describe a record of exactly 33 standard data format characters. Multiple redefinitions of this record are permitted. VALUE clauses for data items not in the first redefinition do not cause those data items to have an initial value when the program is placed into its initial state, but will be used for the INITIALIZE statement with the VALUE phrase. The MCS always references the record according to the data description defined in item 2f of the general rules for Format 3, which begin on page 4-73.
- *data-name-1*, *data-name-2*, *data-name-3*, *data-name-4*, *data-name-5* and *data-name-6* must be unique within the CD entry. Within this series, any data-name may be replaced by the reserved word FILLER.

Input CD General Rules

The following general rules apply to Format 1:

1. The input 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.
2. For each input CD entry, a record area of 87 contiguous character positions 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 through 12 in the record.
   b. The SYMBOLIC SUB-QUEUE-1 clause defines *data-name-2* as the name of an elementary alphanumeric data item of 12 characters occupying positions 13 through 24 in the record.
   c. The SYMBOLIC SUB-QUEUE-2 clause defines *data-name-3* as the name of an elementary alphanumeric data item of 12 characters occupying positions 25 through 36 in the record.

d. The SYMBOLIC SUB-QUEUE-3 clause defines *data-name-4* as the name of an elementary alphanumeric data item of 12 characters occupying positions 37 through 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 six digits, without an operational sign, occupying character positions 49 through 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 eight digits, without an operational sign, occupying character positions 55 through 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 through 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 four digits, without an operational sign, occupying character positions 75 through 78 in the record.

i. The END KEY clause defines *data-name-9* as the name of an elementary alphanumeric data item of one 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 two characters occupying positions 80 and 81 in the record.

k. The MESSAGE COUNT clause defines *data-name-11* as the name of an elementary data item whose implicit description is that of an integer of six digits, without an operational sign, occupying character positions 82 through 87 in the record.

The second option (*data-name-1, data-name-2, . . ., data-name-11*) 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:

```
01  FILLER.
    02  data-name-1          PICTURE X(12).
    02  data-name-2          PICTURE X(12).
    02  data-name-3          PICTURE X(12).
    02  data-name-4          PICTURE X(12).
    02  data-name-5          PICTURE 9(6).
    02  data-name-6          PICTURE 9(8).
    02  data-name-7          PICTURE X(12).
    02  data-name-8          PICTURE 9(4).
    02  data-name-9          PICTURE X.
    02  data-name-10         PICTURE XX.
    02  data-name-11         PICTURE 9(6).
```

3. The contents of data items referenced by *data-name-2*, *data-name-3* and *data-name-4*, when not being used must contain spaces.

4. The data items referenced by *data-name-1*, *data-name-2*, *data-name-3* and *data-name-4* contain symbolic names designating queues, sub-queues, . . ., respectively. These symbolic names must follow the rules for the formation of system-names, and must have been previously defined to the MCS.

5. 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. At the time of execution of a RECEIVE statement, the *data-name-1* field in the input CD area must contain the name of a symbolic queue. The data items specified by *data-name-2*, *data-name-3* and *data-name-4* may contain symbolic sub-queue names or spaces. When a given level of the queue structure is specified, all higher levels must also be specified. If less than all the levels of the queue hierarchy are specified, the MCS determines the next message or portion of a message to be accessed within the queue or sub-queue specified in the input CD. 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.

6. Whenever a program is scheduled by the MCS to process a message, that program establishes a run unit and the symbolic names of the queue structure that demanded this activity 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.

7. If the MCS attempts to schedule a program lacking an INITIAL clause, the results are undefined.

8. During the execution of a RECEIVE statement, the MCS provides, in the data item referenced by *data-name-5*, the date on which it recognized that the message was complete in the form YYMMDD (year, month, day). The contents of the data item referenced by *data-name-5* are not updated by the MCS other than as part of the execution of a RECEIVE statement.

9. During the execution of a RECEIVE statement, the MCS provides, in the data item referenced by *data-name-6*, the time at which it recognized that the message was complete in the form HHMMSShh (hours, minutes, seconds, hundredths of a second). The contents of the data item referenced by *data-name-6* are not updated by the MCS other than as part of the execution of a RECEIVE statement.

10. During the execution of a RECEIVE statement, the MCS provides, in the data item referenced by *data-name-7*, the symbolic name of the communication terminal that is the source of the message being transferred. 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.

11. The MCS indicates through 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.

12. The contents of the data item referenced by *data-name-9* are set by the MCS only as part of the execution of a RECEIVE statement according to the following rules:

    When the RECEIVE MESSAGE phrase is specified:

    a. If an end of group has been detected, the contents of the data item referenced by *data-name-9* are set to 3.

    b. If an end of message has been detected, the contents of the data item referenced by *data-name-9* are set to 2.

    c. If less than a message is transferred, the contents of the data item referenced by *data-name-9* are set to 0.

When the RECEIVE SEGMENT phrase is specified:

a. If an end of group has been detected, the contents of the data item referenced by *data-name-9* are set to 3.

b. If an end of message has been detected, the contents of the data item referenced by *data-name-9* are set to 2.

c. If an end of segment has been detected, the contents of the data item referenced by *data-name-9* are set to 1.

d. If less than a message is transferred, the contents of the data item referenced by *data-name-9* are set to 0.

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*.

13. 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 statement. (See Table 4-6 on page 4-77).

14. The contents of the data item referenced by *data-name-11* indicate the number of messages that exist in a queue, sub-queue-1, and so on. The MCS updates the contents of the data item referenced by *data-name-11* only as part of the execution of an ACCEPT . . . MESSAGE COUNT statement.

Output CD General Rules

The following general rules apply to Format 2:

1. The nature of the output CD information is such that it is not sent to the terminal, but constitutes the communication between the program and the MCS about the message being handled.

2. A record area of contiguous character positions is allocated for each output CD. If the CD entry does not contain a DESTINATION TABLE OCCURS clause, the length of the allocated record area is 23 characters. If the CD entry does contain a DESTINATION TABLE OCCURS clause, the length of the allocated record area is 10 plus 13 times the value of *integer-1*.

The implicit description of the allocated record area is:

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 through 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 four digits, without an operational sign, occupying character positions 5 through 8 in the record.

c. The STATUS KEY clause defines *data-name-3* to be an elementary alphanumeric data item of two characters occupying positions 9 and 10 in the record.

d. Character positions 11 through 23 and every set of 13 characters thereafter will form table items of the following description:

   (1) The ERROR KEY clause defines *data-name-4* as the name of an elementary alphanumeric data item of one character.

   (2) 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 the following:

```
01  FILLER.
    02  data-name-1            PICTURE 9(4).
    02  data-name-2            PICTURE 9(4).
    02  data-name-3            PICTURE XX.
    02  FILLER                 OCCURS integer-1 TIMES.
        03  data-name-4        PICTURE X.
        03  data-name-5        PICTURE X(12).
```

3. During the execution of a SEND, PURGE, ENABLE OUTPUT or DISABLE OUTPUT statement, the content 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*, and so forth, up to and including the occurrence of the area referenced by *data-name-5* that is indicated by the contents of *data-name-1*. If during the execution of a SEND, PURGE, 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-1*, inclusive, an error condition is indicated, no action is taken for any destination, and the execution of the SEND, PURGE, ENABLE OUTPUT or DISABLE OUTPUT statement is terminated. The user must ensure that the value of the data item

referenced by *data-name-1* is valid at the time of the execution of the SEND, PURGE, ENABLE OUTPUT or DISABLE OUTPUT statement.

4. As part of the execution of a SEND statement, the MCS will interpret the content 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 identifier in the SEND statement from which data is to be transferred. (See the discussion of the SEND statement that begins on page 6-173.)

5. Each occurrence of the data item referenced by *data-name-5* contains a symbolic destination name previously known to the MCS. These symbolic destination names must follow the rules for the formation of system-names.

6. The content of the data item referenced by *data-name-3* indicates the status condition of the previously executed SEND, PURGE, ENABLE OUTPUT or DISABLE OUTPUT statement. (See Table 4-6 on page 4-77.)

7. If, during the execution of a SEND, PURGE, ENABLE OUTPUT or DISABLE OUTPUT statement the MCS determines that an error has occurred, the content of the data item referenced by *data-name-3* and all occurrences of the data items referenced by *data-name-4* are updated. The content of the error key data item referenced by *data-name-4*, when nonzero, indicates that an error has occurred for the destination specified by the associated value in the symbolic destination data item referenced by *data-name-5*. Otherwise, the content of the error key data item referenced by *data-name-4* is set to zero. See Table 4-7 on page 4-79 for the meanings of the various error key values.

Input-Output CD General Rules

The following general rules apply to Format 3:

1. 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.

2. 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 six digits, without an operational sign, occupying character positions 1 through 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 eight digits, without an operational sign, occupying character positions 7 through 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 through 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 four digits, without an operational sign, occupying character positions 27 through 30 in the record.

e. The END KEY clause defines *data-name-5* as the name of an elementary alphanumeric data item of one 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 two characters occupying positions 32 and 33 in the record.

The second option (*data-name-1*, *data-name-2*, . . ., *data-name-6*) may be used to replace the above clauses 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:

```
01  FILLER.
    02  data-name-1          PICTURE 9(6).
    02  data-name-2          PICTURE 9(8).
    02  data-name-3          PICTURE X(12).
    02  data-name-4          PICTURE 9(4).
    02  data-name-5          PICTURE X.
    02  data-name-6          PICTURE XX.
```

3. 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.

4. *data-name-1* 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-1* are updated only by the MCS as part of the execution of a RECEIVE statement.

5. *data-name-2* has the format HHMMSShh (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-2* are updated only by the MCS as part of the execution of a RECEIVE statement.

6. Whenever a program is scheduled by the MCS to process a message, that program establishes a run unit and 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 contents of the data item referenced by *data-name-3* of the input-output CD associated with the INITIAL clause are 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.

7. If the MCS attempts to schedule a program lacking an INITIAL clause, the results are undefined.

8. When the INITIAL clause is specified for an input-output CD and the program is scheduled by the MCS, the contents of the data item referenced by *data-name-3* must not be changed by the program. If the contents are changed, the execution of any statement referencing *cd-name* is unsuccessful, and the data item referenced by *data-name-6* is set to indicate unknown source or destination, as applicable.

9. 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.*

   After executing the first statement referencing *cd-name*, the contents of the data item referenced by *data-name-3* must not be changed by the program. If the contents are changed, the execution of any statement referencing *cd-name* is unsuccessful, and the data item referenced by *data-name-6* is set to indicate unknown source or destination, as applicable.

10. The MCS indicates, through the contents of the data item referenced by *data-name-4*, the number of character positions filled as a result of the execution of the RECEIVE statement.

    As part of the execution of a SEND statement, the MCS interprets the contents of the data item referenced by *data-name-4* as 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 transferred.

11. The contents of the data item referenced by *data-name-5* are set only by the MCS as part of the execution of a RECEIVE statement according to the following rules:

    When the RECEIVE MESSAGE phrase is specified:

    a. If an end of group has been detected, the contents of the data item referenced by *data-name-5* are set to 3.

    b. If an end of message has been detected, the contents of the data item referenced by *data-name-5* are set to 2.

    c. If less than a message is transferred, the contents of the data item referenced by *data-name-5* are set to 0.

    When the RECEIVE SEGMENT phrase is specified:

    a. If an end of group has been detected, the contents of the data item referenced by *data-name-5* are set to 3.

    b. If an end of message has been detected, the contents of the data item referenced by *data-name-5* are set to 2.

    c. If an end of segment has been detected, the contents of the data item referenced by *data-name-5* are set to 1.

    d. If less than a message is transferred, the contents of the data item referenced by *data-name-5* are set to 0.

       When more than one of the conditions is satisfied simultaneously, the rule first satisfied in the order listed determines the contents of the data item referenced by *data-name-5*.

12. The contents of the data item referenced by *data-name-6* indicate the status condition of the previously executed PURGE, RECEIVE or SEND statement. (See Table 4-6 on page 4-77.)

Status Key Conditions

Table 4-6 indicates the possible content of the data item referenced by *data-name-10* (Format 1), *data-name-3* (Format 2), or *data-name-6* (Format 3) at the completion of each statement shown. A ▌ symbol on a line in a statement column indicates that the associated code shown for that line is possible for that statement.

Table 4-6: Communication Status Key Conditions

Description	Status Key Value	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
No error detected. Action completed.	00	■	■	■	■	■	■	■	■	■	■	■
One or more destinations disabled. Action completed.	10	□	□	■	■	□	□	□	□	□	□	□
Destination disabled. No action taken.	10	□	■	□	□	□	□	□	□	□	□	□
Symbolic source, or one or more queues or destinations already disabled/enabled.	15	□	□	□	□	□	■	■	■	■	■	■
One or more destinations unknown. Action completed for known destinations.	20	□	■	■	■	□	□	□	■	□	□	■
One or more queues or sub-queues unknown. No action taken.	20	■	□	□	□	■	■	□	□	■	□	□

■ *The status is allowed.*

□ *The status is not allowed.*

Table 4-6: Communication Status Key Conditions *(Cont.)*

Description	Status Key Value	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
Symbolic source is unknown. No action taken.	21	■						■			■	
Destination count invalid. No action taken.	30			■	■				■			■
Password invalid. No enabling/disabling action taken.	40						■	■	■	■	■	■
Text length exceeds size of *identifier-1*.	50		■	■								
Portion requested to be sent has text length of zero or *identifier-1* absent. No action taken.	60		■	■								
Output queue capacity exceeded.	65			■								
One or more destinations do not have portions associated with them. Action completed for other destinations.	70				■							
A combination of at least two status key conditions 10, 15, and 20 have occurred.	80			■	■		■		■	■		■

Error Key Values

Table 4-7 indicates the possible content of the data item referenced by *data-name-4* (Format 2) at the completion of each statement shown. A ▌ symbol on a line in a statement column indicates that the associated error key value shown for that line is possible for that statement.

Table 4-7: Error Key Values

Description	Error Key Value	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT
No error.	0	▌	▌	▌	▌
Symbolic destination unknown.	1	▌	▌	▌	▌
Symbolic destination disabled.	2	▌	▌		
No partial message with referenced symbolic destination.	4		▌		
Symbolic destination already enabled/disabled.	5			▌	▌
Output queue capacity exceeded.	6	▌			
Reserved for future use.	7–9				

▌ *The status is allowed.*

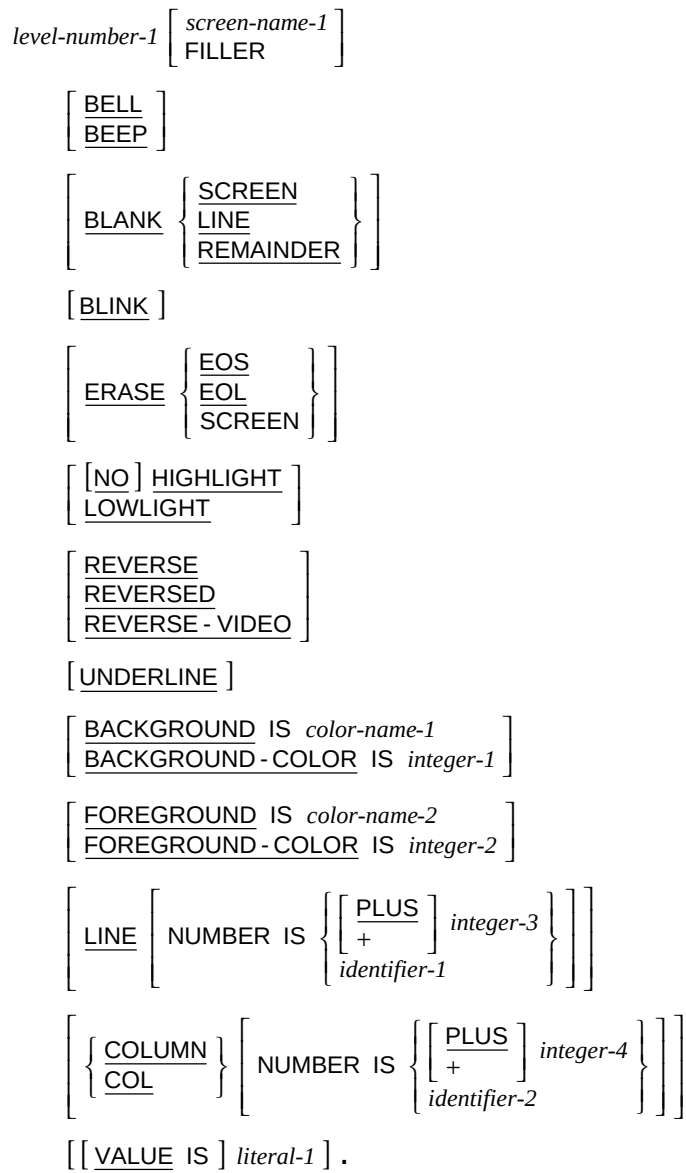
The status is not allowed.

Screen Description Entry

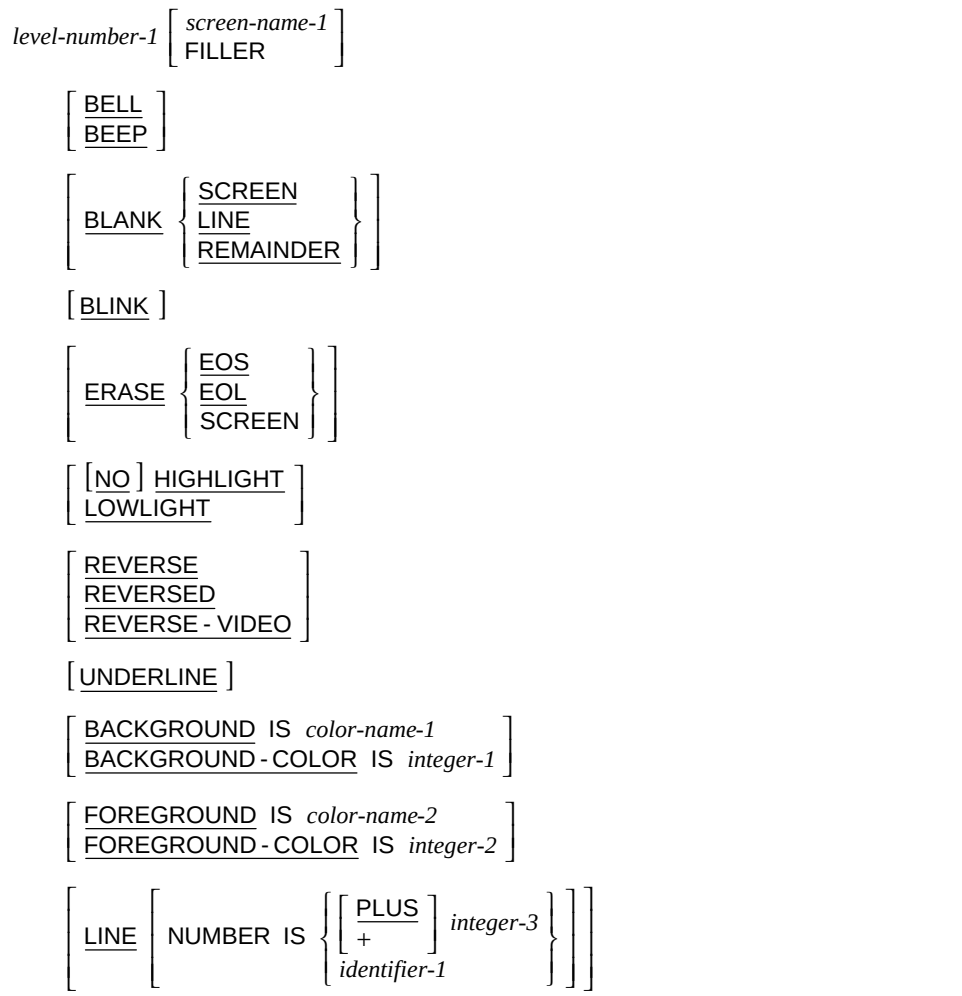
Format 1: Screen Group

```
level-number-1 [ screen-name-1 ]
               [ FILLER      ]

    [ BACKGROUND IS color-name-1 ]
    [ BACKGROUND-COLOR IS integer-1 ]

    [ FOREGROUND IS color-name-2 ]
    [ FOREGROUND-COLOR IS integer-2 ]

    [ [ USAGE IS ] DISPLAY ]

    [ [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
                  { TRAILING }

    [ AUTO ]

    [ SECURE ]

    [ REQUIRED ]

    [ FULL ] .

    { screen-description-entry-1 }...
```

Format 2: Screen Literal

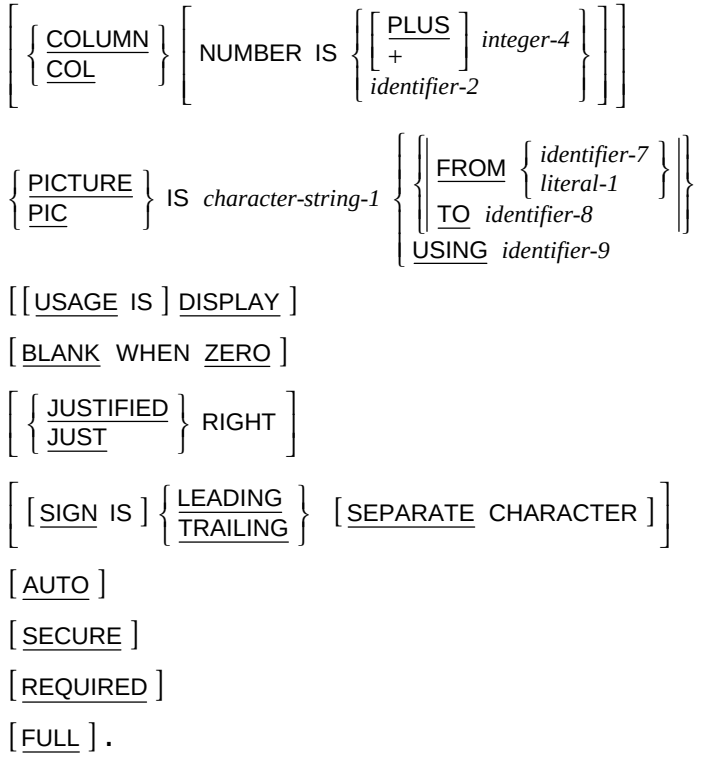


Format 3: Screen Field



(continued on next page)

Format 3: Screen Field *(continued from previous page)*



level-number-1 must be in the range 01 to 49, or 77. Level-numbers 66 and 88 are not allowed. Level-numbers in the range 01 through 49 are used to define group and elementary fields in the same way as in the other sections of the Data Division.

A screen description entry that contains a screen-name following the level-number defines that screen-name. Screen-names may be used only in ACCEPT and DISPLAY statements. A screen-name is not a data-name, and the two types of names are not interchangeable.

A particular screen-attribute may not be specified more than once in a given screen description entry. The possible screen-attributes are defined in the following subsections. Note that a number of the clauses that can be used in other sections of the Data Division are not available in the Screen Section. These include OCCURS, REDEFINES, RENAMES, SIGN, SYNC and USAGE.

AUTO Clause

AUTO

The AUTO clause may be used either at the group level or at the elementary level. When used at the group level the effect is as if it were specified in each subordinate elementary entry that specifies a PICTURE clause with a TO or USING option.

When an elementary field that has the AUTO attribute is functioning as an input field during the course of an ACCEPT operation, the field is considered to be complete as soon as sufficient characters have been entered to fill the field. If the field is not the last input field in the group to which it belongs, the cursor moves to the next field and the ACCEPT operation continues. If the field is the last input field in the group to which it belongs, the ACCEPT operation terminates.

In the absence of the AUTO attribute, the operator must explicitly terminate each field before the cursor moves to the next input field.

BACKGROUND Clause

[BACKGROUND IS *color-name-1*
BACKGROUND-COLOR IS *integer-1*]

color-name-1 may be any properly formed user-defined word that names a color known to the runtime system. The default names known to all RM/COBOL runtime systems are provided in Table 4-8. *color-name-1* is not a data-name, that is, *color-name-1* must be the color-name itself and does not refer to a data item that contains the color-name.

The BACKGROUND clause may be used either at the group level or at the elementary level. When used at the group level the effect is as if it were specified in each subordinate elementary entry that is not controlled by an intervening nested BACKGROUND clause.

The BACKGROUND clause causes the background of the screen field to be shown in the specified color provided the terminal supports color operations and provided the appropriate configuration operations have been performed. The specification is effective for both input and output fields.

The BACKGROUND-COLOR clause is an alternative method of specifying the background color for the screen item. It is provided for compatibility with other common dialects of COBOL. The value of *integer-1* must be in the range 0 through 7 and specifies a color-name according to Table 4-8.

Table 4-8: Color Integers

Value	COLOR-NAME
0	BLACK
1	BLUE
2	GREEN
3	CYAN
4	RED
5	MAGENTA
6	BROWN
7	WHITE

The BACKGROUND-COLOR clause may be used either at the group level or at the elementary level with the same result as a BACKGROUND clause that specifies a color-name corresponding to the value of *integer-1*.

BELL Clause

```
[ BELL ]
[ BEEP ]
```

During the course of a DISPLAY operation, the terminal's audible alert signal is sounded when the cursor encounters an elementary input field that has the BELL attribute.

The BELL clause may be used only at the elementary level.

The words BELL and BEEP are synonymous.

BLANK LINE Clause

```
BLANK LINE
```

During the course of a DISPLAY operation, when the cursor encounters an elementary field that has the BLANK LINE attribute, the line from the position of the cursor to the right end of the line is cleared to spaces. The position of the cursor remains unchanged.

The BLANK LINE clause may be used only at the elementary level.

BLANK REMAINDER Clause

BLANK REMAINDER

During the course of a DISPLAY operation, when the cursor encounters an elementary field that has the BLANK REMAINDER attribute, the line from the position of the cursor to the right end of the line and all lines below the cursor are cleared to spaces. The position of the cursor remains unchanged.

The BLANK REMAINDER clause may be used only at the elementary level.

BLANK SCREEN Clause

BLANK SCREEN

During the course of a DISPLAY operation, when the cursor encounters an elementary field that has the BLANK SCREEN attribute, the entire screen is cleared to spaces. The position of the cursor remains unchanged.

The BLANK SCREEN clause may be used only at the elementary level.

Regardless of the order in which they appear in the screen description entry, the following screen attributes are always acted on in the following order:

1. BLANK SCREEN or ERASE SCREEN
2. LINE or COLUMN positioning
3. BLANK REMAINDER or ERASE EOS
4. BLANK LINE or ERASE EOL

Therefore, it is redundant to specify BLANK LINE in the same entry with BLANK REMAINDER, and it is redundant to specify either of those attributes in the same entry with BLANK SCREEN.

BLANK WHEN ZERO Clause

BLANK WHEN ZERO

The BLANK WHEN ZERO clause has the same effect in the Screen Section as it does in the other sections of the Data Division. It causes the screen field to be filled with spaces if the value of the associated item is zero. It is effective only during an output operation.

The BLANK WHEN ZERO clause can be used only at the elementary level for a screen item whose category is numeric or numeric edited.

The BLANK WHEN ZERO clause must not be specified in the same entry with a PICTURE clause having an asterisk as the zero suppression symbol.

The BLANK WHEN ZERO clause must not be specified in the same entry with a PICTURE clause that specifies an operational sign with the symbol S.

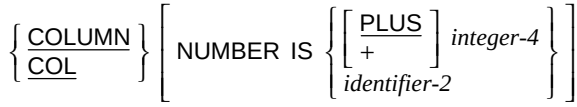
BLINK Clause

BLINK

During both ACCEPT and DISPLAY operations, the BLINK clause causes the screen field to be shown in the flashing mode if such a mode is available on the terminal.

The BLINK clause may be used only at the elementary level.

COLUMN Clause



The COLUMN clause may be used to specify an absolute or relative horizontal position for the cursor. It may be specified only at the elementary level. The words COLUMN and COL are synonymous.

If *identifier-2* is used, it must be defined in one of the other sections of the Data Division as an elementary numeric integer. It may be qualified or subscripted, but reference modification is not permitted.

If *identifier-2* is used or if *integer-4* is used without the PLUS option, the current value of the data item referred to by *identifier-2* or the value of *integer-4* is interpreted as an absolute 1-relative column position. That is, a value of 1 specifies the leftmost column position in the current line. Behavior of the cursor is undefined if the value is less than 1 or greater than the width of the screen.

If the PLUS option is used, the value of *integer-4* is interpreted as an increment to the current position of the cursor, and the cursor is advanced to the right the specified number of positions. Behavior of the cursor is undefined if such advancement moves the cursor beyond the right edge of the screen.

If the COLUMN clause is omitted in an elementary screen description entry, the horizontal cursor position is:

- 1 if the LINE clause is used in the same screen description entry.
- the current cursor position if the LINE clause is also omitted.

See Table 4-9 on page 4-91 for a summary of the interaction of the LINE and COLUMN clauses.

A COLUMN clause with no operand is equivalent to a COLUMN PLUS 1 clause.

ERASE Clause

The ERASE clause may be specified only for elementary screen description entries.

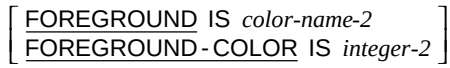
During a display operation, when displaying an elementary field described with the ERASE clause that specifies the EOS option, the line from the position of the cursor to the right end of the line and all lines below the cursor are cleared to spaces. The position of the cursor remains unchanged at the beginning of the screen field.

During a display operation, when displaying an elementary field described with the ERASE clause that specifies the EOL option, the line from the position of the cursor to the right end of the line is cleared to spaces. The position of the cursor remains unchanged at the beginning of the screen field.

During a display operation, when displaying an elementary field described with the ERASE clause that explicitly or implicitly specifies the SCREEN option, the line from the position of the cursor to the right end of the line is cleared to spaces. The position of the cursor remains unchanged at the beginning of the screen field.

See “BLANK SCREEN Clause” on page 4-86 for additional information on blanking (erasing) the screen and portions of the screen.

FOREGROUND Clause



color-name-2 may be any properly formed user-defined word that names a color known to the runtime system. The default names known to all RM/COBOL runtime systems are

provided in Table 4-8 on page 4-85. *color-name-2* is not a data-name, that is, *color-name-2* must be the color-name itself and does not refer to a data item that contains the color-name.

The FOREGROUND clause may be used either at the group level or at the elementary level. When used at the group level the effect is as if it were specified in each subordinate elementary entry that is not controlled by an intervening nested FOREGROUND clause.

The FOREGROUND clause causes the foreground of the screen field to be shown in the specified color provided the terminal supports color operations and provided the appropriate configuration operations have been performed. The specification is effective for both input and output fields.

The FOREGROUND-COLOR clause is an alternative method of specifying the foreground color for the screen item. It is provided for compatibility with other common dialects of COBOL. The value of *integer-2* must be in the range 0 through 7 and specifies a color-name according to Table 4-8 on page 4-85.

The FOREGROUND-COLOR clause may be used either at the group level or at the elementary level with the same result as a FOREGROUND clause that specifies a color-name corresponding to the value of *integer-2*.

FULL Clause

FULL

The FULL clause may be used either at the group level or at the elementary level. When used at the group level the effect is as if it were specified in each subordinate elementary entry that specifies a PICTURE clause with a TO or USING option.

When an elementary field that has the FULL attribute is functioning as an input field during the course of an ACCEPT operation, the user is required to enter either a field terminator by itself, in which case the field is bypassed and the value of the associated item remains unchanged, or a sufficient number of characters to fill the entire screen field. Partially filling the screen field is not allowed. If the REQUIRED attribute is also specified, the option of entering a field terminator by itself is not available.

HIGHLIGHT and LOWLIGHT Clauses

[[NO] HIGHLIGHT
| LOWLIGHT]

An elementary field that is described with the HIGHLIGHT clause is shown at high intensity during both accept and display operations.

An elementary field that is described with the LOWLIGHT clause or NO HIGHLIGHT clause is shown at low intensity during both accept and display operations.

The default intensity is high for accept operations and low for display operations.

The HIGHLIGHT, LOWLIGHT and NO HIGHLIGHT clauses may be specified only in elementary screen description entries.

JUSTIFIED Clause

```
{ JUSTIFIED } RIGHT
{ JUST      }
```

The JUSTIFIED clause has the same effect in the Screen Section as it does in the other sections of the Data Division. That is, it specifies nonstandard positioning of nonnumeric data within the screen field when the screen field is acting as a receiving field. If the associated item is longer than the screen field, the leftmost characters of the associated item are truncated and the remaining characters from the associated item are moved into the screen field. If the associated item is shorter than the screen field, the remaining leftmost positions are space-filled. In either case, the rightmost character from the associated item falls in the rightmost position of the screen field.

The JUSTIFIED clause may be used only at the elementary level and only with screen fields whose category is alphanumeric or alphabetic. It is effective only during ACCEPT operations.

JUSTIFIED and JUST are synonymous.

LINE Clause

```
LINE [ NUMBER IS { [ PLUS ] integer-3 } ]
                 { [ +    ]            }
                 { identifier-1        }
```

The LINE clause may be used to specify an absolute or relative vertical position for the cursor. It may be specified only at the elementary level.

If *identifier-1* is used, it must be defined in one of the other sections of the Data Division as an elementary numeric integer. It may be qualified or subscripted, but reference modification is not permitted.

If *identifier-1* is used or if *integer-3* is used without the PLUS option the current value of the data item referred to by *identifier-1* or the value of *integer-3* is interpreted as an absolute 1-relative line position. That is, a value of 1 specifies the topmost line on the

screen. Behavior of the cursor is undefined if the value is less than 1 or greater than the number of lines available on the screen.

If the PLUS option is used, the value of *integer-3* is interpreted as an increment to the current position of the cursor, and the cursor is undefined if such advancement moves the cursor below the bottom edge of the screen.

If the LINE clause is omitted in an elementary screen description entry, the cursor position remains on the current line.

If the LINE clause is used without the COLUMN clause, the cursor is moved to the leftmost position of the specified line. Table 4-9 shows the interaction of LINE and COLUMN clauses in a screen description entry.

A LINE clause with no operand is equivalent to a LINE PLUS 1 clause.

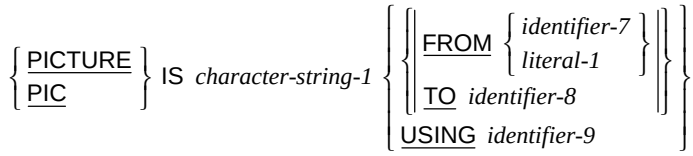
Table 4-9: Interaction of LINE and COLUMN Clauses in a Screen Description Entry

LINE Clause	COLUMN Clause	Field Position (line 1, column 2)
omitted	*omitted*	(*Curline*, *Curcol*)
	COLUMN	(*Curline*, *Curcol* + 1)
	COLUMN *n*	(*Curline*, *n*)
	COLUMN PLUS *n*	(*Curline*, *Curcol* + *n*)
LINE	*omitted*	(*Curline* + 1, 1)
	COLUMN	(*Curline* + 1, *Curcol* + 1)
	COLUMN *n*	(*Curline* + 1, *n*)
	COLUMN PLUS *n*	(*Curline* + 1, *Curcol* + *n*)
LINE *m*	*omitted*	(*m*, 1)
	COLUMN	(*m*, *Curcol* + 1)
	COLUMN *n*	(*m*, *n*)
	COLUMN PLUS *n*	(*m*, *Curcol* + *n*)
LINE PLUS *m*	*omitted*	(*Curline* + *m*, 1)
	COLUMN	(*Curline* + *m*, *Curcol* + 1)
	COLUMN *n*	(*Curline* + *m*, *n*)
	COLUMN PLUS *n*	(*Curline* + *m*, *Curcol* + *n*)

[1] *Curline is 1 for the first elementary entry in a screen record and is the line number of the immediately preceding elementary entry for each subsequent entry in the screen record.*

[2] *Curcol is 1 for the first elementary entry in a screen record and is the column number plus field width of the immediately preceding elementary entry for each subsequent entry in the screen record.*

PICTURE Clause



The PICTURE clause may be specified only at the elementary level; it may not be specified in the same screen description entry as a VALUE clause.

PICTURE and PIC are synonymous.

character-string-1 is defined in the same way and has the same interpretation as in the other sections of the Data Division.

As indicated by the format, a PICTURE clause in the Screen Section must contain either one or two associated items specified in the FROM, TO or USING phrases. Two associated items may be specified only if both a FROM and TO phrase are specified. The USING phrase is equivalent to specifying both a FROM and TO phrase, each of which specify the same *identifier-9*.

When *identifier-7*, *identifier-8*, or *identifier-9* are specified, they must have been defined as data items in one of the other sections of the Data Division. They may not be reference modified, but qualification and subscripting may be used.

When an identifier is specified as an associated data item, the compiler allocates a unique memory area for that screen item to serve as an intermediate storage area for the transmission of data between the screen field and the associated data item. The size of the intermediate storage area is determined by the PICTURE character-string in the same way as for the other sections of the Data Division.

The presence of a FROM or USING phrase in the description of a screen item marks that screen item as an output item that is active during DISPLAY operations. The execution of a DISPLAY statement causes an implicit MOVE from the associated data item to the screen item prior to displaying the screen field.

The presence of a TO or USING phrase in the description of a screen item marks that screen item as an input item that is active during ACCEPT operations. The execution of an ACCEPT statement causes an implicit MOVE from the screen item to the associated data item after accepting the screen field.

REQUIRED Clause

REQUIRED

The REQUIRED clause may be used either at the group level or at the elementary level. When used at the group level the effect is as if it were specified in each subordinate elementary entry that specifies a PICTURE clause with a TO or USING option.

When an elementary field that has the REQUIRED attribute is functioning as an input field during the course of an ACCEPT operation, the user is required to enter at least one character in the field.

REVERSE Clause

[REVERSE
REVERSED
REVERSE-VIDEO]

An elementary field that has the REVERSE attribute is shown in reverse video during both ACCEPT and DISPLAY operations.

The REVERSE clause may be used only at the elementary level.

The words REVERSE, REVERSED, and REVERSE-VIDEO are synonymous.

SECURE Clause

SECURE

The SECURE clause may be used either at the group level or at the elementary level. When used at the group level the effect is as if it were specified in each subordinate elementary entry that specifies a PICTURE clause with a TO or USING option.

When an elementary field that has the SECURE attribute is functioning as an input field during the course of an ACCEPT operation, the characters entered by the user are moved to the intermediate area but are not shown on the screen. Instead, asterisks are placed in the screen field for each character entered by the user.

SIGN Clause

```
[ SIGN IS ] { LEADING  }   [ SEPARATE CHARACTER ]
            { TRAILING }
```

The SIGN clause in a screen description entry has the same function and rules as in other sections of the Data Division, except that for screen items, the operational sign is always separate. The SIGN clause may be specified in either a group screen description entry or an elementary field screen description entry. When used at the group level it applies to all elementary items subordinate to that group that are not subordinate to an intervening nested SIGN clause.

UNDERLINE Clause

```
UNDERLINE
```

An elementary field that has the UNDERLINE attribute is shown in underline mode during both ACCEPT and DISPLAY operations, provided the terminal supports that mode.

The UNDERLINE clause may be used only at the elementary level.

USAGE Clause

```
[ USAGE IS ] DISPLAY
```

The USAGE clause may be used in either a group screen description entry or an elementary field screen description entry. When used at the group level, it applies to all elementary items subordinate to that group.

The USAGE clause in the Screen Section can specify only DISPLAY usage. DISPLAY usage indicates that the format of the data is a standard data format. 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

[VALUE IS] *literal-1*

literal-1 must be a nonnumeric literal.

The VALUE clause may be used only at the elementary level. It may not be specified in the same screen description entry as a PICTURE, BLANK WHEN ZERO, JUSTIFIED, SIGN, USAGE, SECURE, AUTO, REQUIRED or FULL clause.

Screen fields whose description includes a VALUE clause are active during DISPLAY operations.

Data Structures

Classes of Data

The five categories of data items (see "PICTURE Clause" on page 4-34) are grouped into three classes:

1. **Alphabetic**
2. **Numeric**
3. **Alphanumeric**

For alphabetic 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 both 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 that group item.

Table 4-10 depicts the relationship of the class and categories of data items.

Table 4-10: Data Item Relationships

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric
Nonelementary (Group)	Alphanumeric	Alphabetic Numeric Numeric Edited Alphanumeric Edited Alphanumeric

Standard Alignment Rules

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

If the receiving data item is described as numeric:

- 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.
- 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 described above.

If the receiving data item is a numeric edited data item, the data moved to the edited data item is aligned by decimal point with zero-fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeroes.

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.

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified as described in the JUSTIFIED clause.

Uniqueness of Reference

Every user-defined name in a COBOL program is assigned, by the user, to name a resource which is to be used in solving a data processing problem. In order to use a resource, a statement in a COBOL program must contain a reference which uniquely identifies that resource. In order to ensure uniqueness of reference, a user-defined name may be qualified, subscripted or reference-modified as described in the following paragraphs.

When the same name has been assigned in separate programs to two or more occurrences of a resource of a given type, and when qualification by itself does not allow the reference in one of those programs to differentiate between the two identically named resources, then certain conventions which limit the scope of names apply. These conventions ensure that the resource identified is that described in the program containing the reference.

Unless otherwise specified by the rules for a statement, any subscripting and reference modification are evaluated only once as the first operation of the execution of that statement.

Qualification

Every user-defined name explicitly referenced in a COBOL source program must be uniquely referenced because either:

- No other name has the identical spelling and hyphenation.
- It is unique within the context of a REDEFINES clause.
- The name exists within a hierarchy of names such that reference to the name can be made unique by mentioning one or more of the higher level names in the hierarchy.

  These higher level names are called qualifiers and the process that specifies uniqueness is called qualification. Identical user-defined names may appear in a source program; however, uniqueness must then be established through qualification for each user-defined name explicitly referenced, except in the case of redefinition. All available qualifiers need not be specified so long as uniqueness is established. Reserved words naming the special registers require qualification to provide uniqueness of reference whenever a source program would result in more than one occurrence of any of these special registers. A paragraph-name or section-name appearing in a program may not be referenced from any other program.
- A program is contained within a program or contains another program.

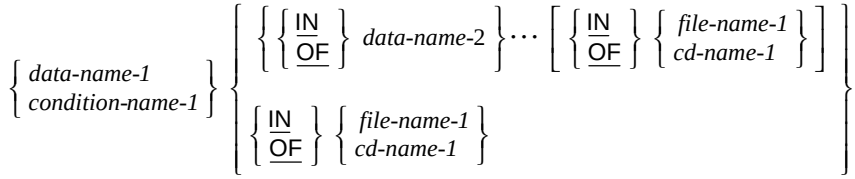
Regardless of the above, the same data-name must not be used as the name of an external record and as the name of any other external data item described in any program contained within or containing the program which describes that external data record. The same data-name must not be used as the name of an item possessing the global attribute and as the name of any other data item described in the program which describes that global data item.

An exception regarding the qualification requirement is made with respect to the operand of a REDEFINES clause because its position within the hierarchical structure of the Data Division implicitly supplies any qualification that might be needed.

In the hierarchy of qualification, names associated with a level indicator are the most significant, followed by names associated with level-number 01, followed by names associated with level-numbers 02, . . ., 49. The name of a conditional variable may be used as a qualifier for any of its condition-names.

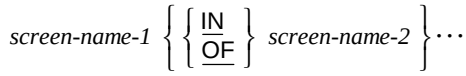
Qualification is performed by following a data-name, condition-name, LINAGE-COUNTER, screen-name, split-key-name, or by one or more phrases made up of a qualifier preceded by IN or OF. IN and OF are logically equivalent.

Format 1: Qualification for Data-Names and Condition-Names



Format 2: Qualification for LINAGE-COUNTER

Format 3: Qualification for Screen-Names



Format 4: Qualification for Split-Key-Names

split-key-name-1 { IN / OF } *file-name-3*

The rules for qualification are as follows:

1. Each reference in the Environment Division, the Data Division, or the Procedure Division to a nonunique user-defined word must be made unique by supplying a sequence of qualifiers that precludes any ambiguity of reference. An exception exists for paragraph-names referenced from the section in which they are defined within the Procedure Division as explained in the section "Procedure References" on page 5-7.
2. *data-name-1* and *data-name-2* may be record-names.
3. Each qualifier must be a name associated with a level indicator (FD, SD or CD), a record-name (level 01), the name of a group item to which the item being qualified is subordinate or the name of the conditional variable with which the condition-name is associated. All qualifiers must be within the same hierarchy as the name being qualified, and they must be specified in the order of successively more inclusive (higher) levels in the hierarchy.
4. The same name must not appear at two levels in a hierarchy.
5. A data-name cannot be subscripted when it is being used as a qualifier.
6. In a program that contains more than one LINAGE clause, each reference to LINAGE-COUNTER must be qualified by the associated file-name.
7. Each reference to a split-key-name, defined in a RECORD KEY or ALTERNATE RECORD KEY clause, must be qualified by the file-name of the file with which the split-key-name is associated if the split-key-name is not unique within the program.
8. A name can be qualified even though it does not need qualification: if there is more than one combination of qualifiers that ensures uniqueness, any such set can be used. Qualified data-names may have any number of qualifiers up to a limit of 49.

Subscripting

Subscripts are used when reference is made to an individual element within a table of like elements that have not been assigned individual data-names.

$$\left\{ \begin{array}{l} \textit{data-name-1} \\ \textit{condition-name-1} \end{array} \right\} \textbf{(} \{ \textit{subscript-1} \}\cdots \textbf{)}$$

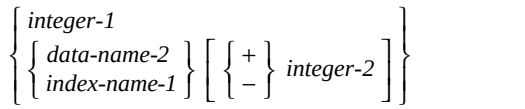
Except as the subject of a SEARCH statement, in a REDEFINES clause, or in a KEY IS phrase of an OCCURS clause, every reference to a table element must be subscripted, and there must be within the parentheses exactly as many subscripts as there are controlling OCCURS clauses for the data item referred to by *data-name-1* or the conditional variable associated with *condition-name-1*.

A data item is controlled by an OCCURS clause if the OCCURS clause is in the data description of the data item or in the data description of a higher-level data item to which the data item is subordinate. A table element is a data item that has at least one controlling OCCURS clause.

Each subscript in the list is associated with a specific OCCURS clause that appears either in the data description of *data-name-1* itself or at a higher level within the same hierarchy. When there is more than one subscript in the parenthesized list, the subscripts are written in the order of successively less inclusive dimensions of the table. That is, the rightmost subscript in the list is associated with the OCCURS clause that appears in the data description of *data-name-1* itself, or the nearest preceding OCCURS clause, if the data description of *data-name-1* does not contain an OCCURS clause.

The value of each subscript is an occurrence number. The lowest possible occurrence number is 1, and an occurrence number of 1 refers to the first element of the table. Higher occurrence numbers (2, 3, . . .) refer in sequence to the following elements of the table. The highest permissible occurrence number for any given dimension of the table is the maximum number of occurrences of the item as specified in the associated OCCURS clause.

The syntax for each individual subscript is:



integer-1 may be signed, but only with a plus sign. When the *integer-1* form of a subscript is used, the occurrence number is the value of *integer-1*.

When the *data-name-2* form of a subscript is used, *data-name-2* may be qualified but not subscripted. It must be defined in the Data Division as a numeric integer data item. The value of the occurrence number of the subscript is the current value of the data item referred to by *data-name-2* optionally incremented (when the + is used) or decremented (when the – is used) by the value of *integer-2*. The value of *integer-2* may be zero. Note that when the *integer-2* option is present, the sum (or difference) of the current value of the data item and *integer-2* must be a valid occurrence number. The user is responsible for ensuring that the current value of the data item referred to by *data-name-2* is appropriate for this use of *data-name-2*. The value of the data item can be modified by a number of different statements such as the MOVE statement, arithmetic statements, and so forth.

When the *index-name-1* form of a subscript is used, the OCCURS clause that is associated with the subscript must specify an INDEXED BY phrase, and *index-name-1* must be defined in the list of that INDEXED BY phrase.

The value of the occurrence number of the subscript is the occurrence number contained in the index referred to by *index-name-1* optionally incremented (when the + is used) or decremented (when the – is used) by the value of *integer-2*. The value of *integer-2* may be zero. Note that when the *integer-2* option is present, the sum (or difference) of the current value of the index and *integer-2* must be a valid occurrence number. The user is responsible for ensuring that the current value of the index referred to by *index-name-1* is appropriate for this use of *index-name-1*. The value of an index can be modified only by the SET statement and by certain forms of the PERFORM and SEARCH statements.

When it is convenient to do so, the *integer-1* or *index-name-1* form of a subscript should be used in preference to the *data-name* form, for efficiency.

Reference Modification

Reference modification permits reference to a subfield of a data item. It may be used anywhere an identifier referencing an alphanumeric data item is allowed, unless explicitly disallowed by the rules for a specific statement.

data-name-1 **(** *leftmost-character-position-1* **:** [*length-1*] **)**

data-name-1 must refer to a data item whose usage is DISPLAY (called "the operand" in this discussion). It may be qualified, subscripted or both.

leftmost-character-position-1 and *length-1* are both arithmetic expressions as defined in the discussion beginning on page 5-20. The value of *leftmost-character-position-1* specifies the leftmost character position of the subfield within the operand. The value of *length-1* specifies the character length of the subfield. The subfield selected in this way is treated as an elementary data item without the JUSTIFIED clause. It has the same class and category as the operand, except that the categories numeric, numeric edited and alphanumeric edited are treated as class and category alphanumeric.

If the operand is described as numeric, numeric edited, alphabetic or alphanumeric edited, it is operated on for purposes of reference modification as if it were redefined as an alphanumeric data item of the same size as the operand.

Each character position of the operand is assigned an ordinal number starting with one at the leftmost character position and incrementing by one for each subsequent character position up to and including the rightmost character position. If the data description for the operand contains or is subject to a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number in the same way as for the digit positions of the data item.

The value of *leftmost-character-position-1* specifies the ordinal position of the leftmost character of the subfield with respect to the leftmost position of the operand. The evaluation must result in a positive integer not greater than the number of characters in the operand.

When *length-1* is omitted, the subfield extends from the position specified by *leftmost-character-position-1* up to and including the rightmost character position of the operand.

When *length-1* is present, its value specifies the length in characters of the subfield. The evaluation must result in a positive integer. The sum of the values of the two expressions minus 1 must not be greater than the number of characters in the operand.

If subscripting is specified for the operand, the reference modification expressions are evaluated immediately after evaluation of the subscripts. If subscripting is not specified

for the operand, the reference modification expressions are evaluated at the time subscripts would have been evaluated had they been specified.

When an identifier that refers to a level-number 01 or 77 Linkage Section data item formal argument is reference modified, the data item is resolved according to its description in the calling program. This is an exception to the rule that formal arguments are resolved according to their description in the Linkage Section of the called program. How the data item is resolved mainly affects the length of the data item as seen in the called program. See the "Linkage Section" topic, which begins on page 4-19, for additional information on this special case of resolving Linkage Section record-names.

Identifier

An identifier is a term used to reflect that a data-name may be followed by a syntactically correct combination of qualifiers, subscripts or reference modifiers to ensure uniqueness.

data-name-1 [{ IN / OF } *data-name-2*]... [{ IN / OF } { *file-name-1* / *cd-name-1* }]

[**(** { *subscript-1* }... **)**] [**(** *leftmost-character-position-1* **:** [*length-1*] **)**]

Condition-Name

Each reference to a condition-name must be unique, or be made unique through qualification, subscripting, or both.

condition-name-1 [{ IN / OF } *data-name-1*]... [{ IN / OF } { *file-name-1* / *cd-name-1* }]

[**(** { *subscript-1* }... **)**]

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 subscripting, references to any of its condition-names also require the same subscripting.

The restrictions on the combined use of qualification and subscripting of condition-names are the same as those that apply to an identifier.

In the general formats, *condition-name-1* refers to a condition-name qualified or subscripted, as necessary for unique reference.

Table Handling

Tables of data are common components of business data processing problems. Although the repeating items that make up a table could be otherwise described as a series of separate data description entries all having the same level-number and all subordinate to the same group item, there are two reasons why this approach is not satisfactory.

First, from a documentation standpoint, the underlying homogeneity of the items would not be readily apparent; second, the problem of making available an individual element of such a table would be severe when there is a decision as to which element is to be made available at object time.

Tables of data items are defined by including the OCCURS clause in their data description entries. This clause specifies that the item is to be repeated as many times as stated. The item is considered to be a table element and its name and description apply to each repetition or occurrence. Since each occurrence of a table element does not have assigned to it a unique data-name, reference to a desired occurrence may be made only by specifying the identifier of the table element together with the occurrence number of the desired table element. The occurrence number is specified by a subscript.

The number of occurrences of a table element may be specified to be fixed or variable depending on the value of another data item.

Table Definition

To define a one-dimensional table, use an OCCURS clause as part of the data description of the table element, but the OCCURS clause must not appear in the description of group items which contain the table element.

Example 1

```
01 TABLE-1.
   02 TABLE-ELEMENT  OCCURS 20 TIMES.
      03 NAME        PICTURE X(40).
      03 SSN         PICTURE 9(9) PACKED-DECIMAL.
```

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table gives rise to a two-dimensional table. To define a two-dimensional table, then, an OCCURS clause must appear in the data description of the element of the table, and in the description of only one group item which contains that table. In the description of a three-dimensional table, the OCCURS clause should appear in the data description of two nested group items which contain the element. The process of nesting table definitions may be continued to any depth, but the size of the outermost group increases geometrically with each dimension.

The following example shows how a three-dimensional table may be defined.

Example 2

```
01 CENSUS-TABLE.
   05 STATE-TABLE               OCCURS 50 TIMES.
      10 STATE-CODE              PIC X(02).
      10 COUNTY-TABLE            OCCURS 50 TIMES.
         15 COUNTY-CODE          PIC X(02).
         15 CITY-TABLE           OCCURS 30 TIMES.
            20 CITY-CODE         PIC X(02).
            20 CITY-POPULATION   PIC 9(07).
```

The data item named STATE-TABLE is a one-dimensional table. The data item named COUNTY-TABLE, which is subordinate to STATE-TABLE, is a two-dimensional table. The data item named CITY-TABLE, which is subordinate to COUNTY-TABLE, is a three-dimensional table.

Example 2 defines 230,101 data items having a total size of 680,100 characters. See Table 4-11.

Table 4-11: Example 2 Definitions

Name	Number	Size
CENSUS-TABLE	1	680100
STATE-TABLE	50	680100
STATE-CODE	50	2
COUNTY-TABLE	2500	13600
COUNTY-CODE	2500	2
CITY-TABLE	75000	270
CITY-CODE	75000	2
CITY-POPULATION	75000	7

References to Table Items

Whenever the user refers to a table element or a condition-name associated with a table element, the reference must indicate which occurrence of the element is intended. For access to a one-dimensional table, the occurrence number of the desired element provides complete information. For access to tables of more than one dimension, an occurrence number must be supplied for each dimension of the table. In Example 2 then, a reference to the fourth STATE-CODE would be complete, whereas a reference to the fourth COUNTY-CODE would not. To refer to COUNTY-CODE, which is an element of a two-dimensional table, the user must refer to, for example, the fourth COUNTY-CODE within the sixth STATE-TABLE.

Occurrence numbers are specified by appending one or more subscripts to the data-name.

The subscript can be represented by an integer, a data-name that references an integer numeric elementary item, or an index-name associated with the table. A data-name or index-name may be followed by either the operator + or the operator – and an integer, which is used as an increment or decrement, respectively. It is permissible to mix integers, data-names and index-names.

The subscripts, enclosed in parentheses, are written immediately following any qualification for the name of the table element. The number of subscripts in such a reference must equal the number of dimensions in the table whose element is being referenced. That is, there must be a subscript for each OCCURS clause in the hierarchy containing the data-name including the data-name itself.

When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization. If a multidimensional table is thought of as a series of nested tables and the most inclusive or outermost table in the nest is considered to be the major table with the innermost or least inclusive table being the minor table, the subscripts are written from left to right in the order major, intermediate and minor.

A reference to an item must not be subscripted if the item is not a table element or an item or condition-name within a table element.

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

When an integer or data-name is used to represent a subscript, it may be used to reference items within different tables. These tables need not have elements of the same size. The same integer or data-name may appear as the only subscript with one item and as one of two or more subscripts with another item.

In order to facilitate such operations as table searching and manipulating specific items, a technique called indexing is available. To use this technique, the programmer assigns one or more index-names to an item whose data description entry contains an OCCURS clause. An index associated with an index-name acts as a subscript, and its value corresponds to an occurrence number for the item with which the index-name is associated.

The INDEXED BY phrase, by which the index-name is identified and associated with its table, is an optional part of the OCCURS clause. There is no separate entry to describe the index associated with index-name since its definition is completely hardware oriented. At object time the contents of the index correspond to an occurrence number for that specific dimension of the table with which the index is associated. The initial value of an index at object time is undefined, and the index must be initialized before use. The initial value of an index is assigned with the PERFORM statement with the VARYING phrase, the SEARCH statement with the ALL phrase, or the SET statement.

The use of an integer or data-name as a subscript referencing a table element does not cause the alteration of any index associated with that table.

An index-name can be used to reference only the table with which it is associated through the INDEXED BY phrase.

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

Relative indexing is an additional option for making references to a table element or to an item within a table element. When the name of a table element is followed by a subscript of the form (index-name + or – integer), the occurrence number required to complete the reference is the same as if index-name were set up or down by integer using the SET statement before the reference. The use of relative indexing does not cause the object program to alter the value of the index.

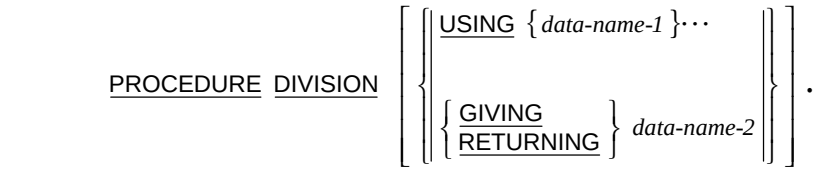
The value of an index can be made accessible to an object program by storing the value in an index data item. Index data items are described in the program by a data description entry containing a USAGE IS INDEX clause. The index value is moved to the index data item by the execution of a SET statement.

Chapter 5: Procedure Division

The Procedure Division contains the procedures to be executed by the object program. It is an optional division within a source program, and it may be omitted if there are no procedures to be executed. The procedures within the Procedure Division may be subdivided into declarative and nondeclarative procedures.

Procedure Division Header

When it is present, the Procedure Division is identified by and must begin with the following header:



The USING phrase of the Procedure Division header identifies the names used by the program for any parameters passed to it by a calling program or from the RM/COBOL Runtime Command. It is required only in one of the following circumstances:

- If the object program is to be invoked by a CALL statement and that statement includes a USING phrase.
- If the object program is to function as the first program in its run unit and it requires access to the text of the runtime command that invokes the object program.

In the first case, the parameters passed to the called program are identified in the USING phrase of the calling program's CALL statement. The correspondence between the two lists of names is established on a positional basis. That is, the first *data-name-1* in the USING list of the CALL statement in the calling program corresponds to the first *data-name-1* in the USING list of the Procedure Division header in the called program, the second *data-name-1* in the USING list of the CALL statement in the calling program corresponds to the second *data-name-1* in the USING list of the Procedure Division header in the called program, and so forth. The two lists need not have the same number of operands, but operands for which there is no corresponding operand in the other list may not be referred to, nor may any of their subordinate data items, condition-names or index-names be referred to in the called program.

In the second case, there is only a single parameter, and it must be defined in the Linkage Section with entries similar to the following:

```
01 MAIN-PARAMETER.
   02 PARAMETER-LENGTH  PIC S9(4) BINARY (2).
   02 PARAMETER-TEXT.
      03 PARAMETER-CHAR PIC X OCCURS 0 TO 100 TIMES
               DEPENDING ON PARAMETER-LENGTH.
```

The Procedure Division header should have the following form:

```
PROCEDURE DIVISION USING MAIN-PARAMETER.
```

Subscripted references to PARAMETER-CHAR can then be used to access the characters within the invocation line. There is further information in the *RM/COBOL User's Guide* describing the technique for passing a character-string to the first program in a run unit.

In either case, each *data-name-1* and *data-name-2* must be defined as a level 01 entry or a level 77 entry in the Linkage Section of the Data Division. A particular user-defined word may not appear more than once as *data-name-1* or *data-name-2*. The data description entry for *data-name-1* or *data-name-2* must not contain a REDEFINES clause. However, *data-name-1* or *data-name-2* may be the object of a REDEFINES clause elsewhere in the Linkage Section.

If the reference to the corresponding data item in the CALL statement declares the parameter to be passed by content, the value of the item is moved when the CALL statement is executed and placed into a system-defined storage item possessing the attributes declared in the Linkage Section for *data-name-1*. The data description of each parameter in the BY CONTENT phrase of the CALL statement must be congruent to the data description of the corresponding parameter in the USING phrase of the Procedure Division header. Two data descriptions are congruent if they specify the same size and, for numeric items, the same usage, scale and sign convention. For binary data items, congruency also depends on both items being allocated with the same number of bytes, which depends on specification of the same binary allocation override in the USAGE clause or matching configuration of the compiler with the BINARY-ALLOCATION and BINARY-ALLOCATION-SIGNED keywords of the COMPILER-OPTIONS configuration record when compiling each of the possibly separately compiled programs.

If the reference to the corresponding data item in the CALL statement declares the parameter to be passed by reference, the object program operates as if the data item in the called program occupies the same storage area as the corresponding data item in the calling program. If *data-name-1* or any of its subordinate elements is referred to in the Procedure Division, it must not be defined as being longer than its corresponding data item in the calling program.

RETURNING is a synonym for GIVING.

The GIVING phrase identifies the name used by the program for a result value. It is required only when the calling program specifies a GIVING phrase in the CALL statement and the design of that calling program depends on a result being placed in its GIVING argument. That is, it is not an error if the calling program specifies a GIVING operand and the called program does not; in this case, the called program ignores and cannot affect the GIVING operand in the calling program. However, it is a data reference error to refer to *data-name-2*, or any of its subordinate data-names or condition-names, redefined data-names, or renamed data-names if the calling program does not have a GIVING phrase, except in certain contexts. When the calling program does not have a GIVING phrase, the contexts where an error will not occur are:

- The ADDRESS OF special register, which will be equal to NULL in this case.
- The USING or GIVING phrase of a CALL statement. In this case, the called program will receive an actual argument with a NULL base address. The called program would cause a data reference error if that program referred to the corresponding formal argument other than in the contexts described here.

Each reference to *data-name-1* or *data-name-2* in the called program is resolved in accordance with the description of *data-name-1* or *data-name-2* as given in the Linkage Section of the called program, except when they name a level 01 or level 77 entry and they are specified in the USING or GIVING phrase of a CALL statement, or when they are reference modified. In these two exception cases, the reference to *data-name-1* or *data-name-2* is resolved according to the description in the calling program.

A data item defined in the Linkage Section of the called program may be referred to within the Procedure Division of that program only if it satisfies one of the following conditions:

- It is an operand in the USING phrase of the Procedure Division header and there is a corresponding operand in the USING phrase of the CALL statement of the calling program.
- It is an operand in the GIVING phrase of the Procedure Division header and there is a corresponding operand in the GIVING phrase of the CALL statement of the calling program.
- It is the target of a Format 5 SET statement using the SET ADDRESS OF *data-name-1* format, where *data-name-1* is the name of the data item and the sending pointer value is not a null pointer value.
- It is subordinate to a data item that satisfies one of the preceding conditions.
- It is defined with a REDEFINES or RENAMES clause, the object of which satisfies one of the preceding conditions.
- It is an item subordinate to an item that satisfies the preceding condition.

- It is a condition-name or index-name associated with a data item that satisfies any of the preceding conditions.
- It is the operand of an ADDRESS special register. In this case, if none of the preceding conditions are satisfied, the ADDRESS special register will have a null pointer value. Thus, an IF statement may be used to verify that one of the preceding conditions has been satisfied by verifying that the ADDRESS of the data item is not equal to NULL.
- It is the operand of a USING or GIVING phrase in a CALL statement. In this case, if none of the preceding conditions are satisfied, the program called by the CALL statement receives an omitted argument for its corresponding formal argument.

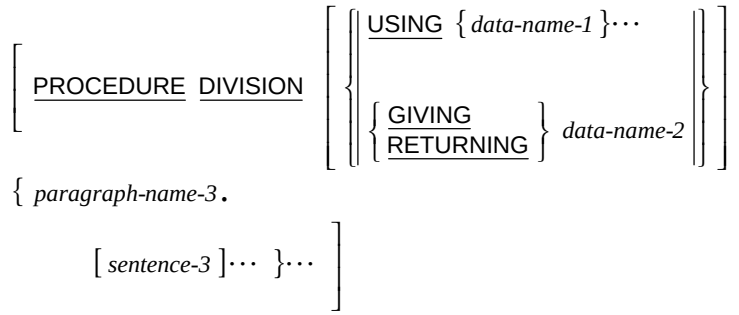
Procedure Division Structure

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

Format 1: Declaratives or Sections

```
[ PROCEDURE DIVISION [ { [ USING { data-name-1 }... ]
                       [ { GIVING    } data-name-2 ] } ] .
                       { RETURNING }
[ DECLARATIVES.
{ section-name-1 SECTION [ segment-number-1 ].
      USE-statement-1.
[ paragraph-name-1.
      [ sentence-1 ]... ]... }...
  END DECLARATIVES. ]
{ section-name-2 SECTION [ segment-number-2 ].
[ paragraph-name-2.
      [ sentence-2 ]... ]... }... ]
```

Format 2: Paragraphs



segment-number-1 must be an integer ranging in value from 0 through 49.

segment-number-2 must be an integer ranging in value from 0 through 127.

If a segment-number is omitted from the section header, the segment-number is assumed to be 0.

All sections that have the same segment-number constitute a program segment. Sections that have the same segment-numbers need not be physically contiguous in the source program.

Segments with segment-number 0 through 49 belong to the fixed portion of the object program.

Segments with segment-number 50 through 127 are independent segments. A program without declaratives may consist solely of independent segments.

Declarative sections must be grouped at the beginning of the Procedure Division, preceded by the keyword DECLARATIVES and followed by the keywords END DECLARATIVES.

Procedures

A procedure comprises a paragraph, a group of successive paragraphs, a section or a group of successive sections within the Procedure Division. If one paragraph is in a section, all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section. It consists of a section-name, a paragraph-name, or a paragraph-name qualified by a section-name.

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

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

In a Procedure Division that is not divided into sections, a paragraph-name may be defined more than once. In a Procedure Division that is divided into sections, a paragraph-name may be defined more than once in the same section. Such nonunique paragraph-names may not be referenced.

A statement is a syntactically valid combination of words and symbols beginning with a verb. The word THEN may be used as a statement separator within the Procedure Division. It has no effect on the meaning of the statements.

Execution

Execution begins with the first statement of the Procedure Division, excluding declaratives.

Statements are then executed in the order in which they appear in the source program, except where the rules indicate some other order.

Procedure References

A procedure is referred to by its paragraph-name or section-name. Paragraph-names may be qualified by the section-name of the section containing the paragraph, whether or not it needs qualification. When referring to a section-name or when using a section-name as a qualifier, the word SECTION must not appear. Qualification is performed by following a paragraph-name with a section-name preceded by IN or OF. IN and OF are synonymous in this context. The general format for paragraph qualification is:

paragraph-name-1 $\left\{ \begin{matrix} \underline{\text{IN}} \\ \underline{\text{OF}} \end{matrix} \right\}$ *section-name-1*

paragraph-name-1 need not be qualified when referred to within the section in which it is defined or when it is unique.

Explicit and Implicit Transfers of Control

The mechanism that controls 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 the writing of an explicit Procedure Division statement, and, therefore, is an implicit transfer of control.

RM/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.

RM/COBOL provides the following types of implicit control flow alterations which override the statement-to-statement transfers of control:

- If a paragraph is being executed under control of another statement (PERFORM, USE, SORT or MERGE), and the paragraph is the last paragraph in the range of the controlling statement, an implied transfer of control occurs from the last statement in the paragraph to the control mechanism of the last-executed controlling statement.
- If a paragraph is being executed under the control of a PERFORM statement that causes iterative execution, and that paragraph is the first paragraph in the range of

that PERFORM statement, an implicit transfer of control occurs between the control mechanism associated with that PERFORM statement and the first statement in that paragraph for each iterative execution of the paragraph.

- When a SORT or MERGE statement is executed, an implicit transfer of control occurs to any associated input or output procedures.

- When any statement is executed that 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 back to the statement that caused the execution of the declarative.

An explicit transfer of control consists of an alteration of the implicit control transfer mechanism by the execution of a procedure branching or conditional statement. An explicit transfer of control can be caused only by the execution of a procedure branching or conditional statement. The execution of the procedure branching ALTER statement 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 only when the statement is executed in a called program.

The term “next executable statement” refers to the next statement to which control is transferred according to the rules above and the rules associated with each language element in the Procedure Division.

There is no next executable statement following:

- The last statement in a declarative section when the paragraph in which it appears is not being executed under the control of some other statement.

- The last statement in a declarative section when the statement is in the range of an active PERFORM statement executed in a different section and this last statement of the declarative section is not also the last statement of the procedure that is the exit of the active PERFORM statement.

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

- A STOP RUN statement or EXIT PROGRAM statement that transfers control outside the program.

- The end program header.

There is also no next executable statement when the program contains no Procedure Division.

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

Segmentation

Segmentation allows the user to segment the Procedure Division of a program, and to specify overlays among the segments. Thus, less runtime memory is required to execute the program. There is no provision for segmenting the data regions of a program.

Segments

When segmentation is used, the Procedure Division must be written as a series of 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 as determined by the assignment of segment-numbers. All paragraphs that contain the same segment-number in their section headers are considered at object time to be one segment. Since segment-numbers can range from 0 through 127, it is possible to subdivide the object program into a maximum of 128 segments. Segmentation has no effect on the need to qualify procedure-names to ensure uniqueness.

Fixed Portion

The fixed portion of the object program is logically treated as if it were always in memory. All sections whose segment-number is less than 50 belong to the fixed portion. The fixed portion of the program is made up of two types of segments: fixed permanent segments and fixed overlayable segments.

A fixed permanent segment is a segment in the fixed portion that 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 so as to reduce memory utilization. Such a segment, if called for by the program, is always made available in its last-used state. Variation of the number of fixed permanent segments in the fixed portion can be accomplished by using the SEGMENT-LIMIT clause.

Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlaid by, a fixed overlayable segment or another independent segment. An independent segment has a segment-number of 50 through 127.

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 the program.

On subsequent transfers of control to the segment, an independent segment is also in its initial state when:

- 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.
- 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.
- Control is transferred explicitly to that segment from a segment with a different segment-number.

On subsequent transfers of control to the segment, an independent segment is in its last-used state when:

- Control is transferred implicitly to that segment from a segment with a different segment-number (except as noted previously).
- 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.
- 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:

- Logic Requirements. Sections that must be available for reference at all times, or are referred to very frequently, are normally classified as belonging to one of the permanent segments; sections that are used less frequently are normally classified as belonging either to one of the overlayable fixed segments or to one of the independent segments, depending on logic requirements.
- Frequency of Use. Generally, the more frequently a section is referred to, the lower its segment-number; the less frequently it is referred to, the higher its segment-number.
- Relationship to Other Sections. Sections which 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; that is, it is not mandatory to transfer control to the beginning of a section.

Restrictions on Program Flow

When segmentation is used, the following restrictions are placed on the ALTER, PERFORM, MERGE and SORT statements.

ALTER Statement Restrictions

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.

PERFORM Statement Restrictions

A PERFORM statement in the fixed portion can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

- Sections or paragraphs wholly contained in the fixed portion.
- Sections or paragraphs wholly contained in a single independent segment.

A PERFORM statement 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 or paragraphs wholly contained in the fixed portion.
- Sections or paragraphs wholly contained in the same independent segment as that PERFORM statement.

MERGE Statement Restrictions

If a MERGE statement appears in the fixed portion, any output procedure referenced by that MERGE statement must be entirely within the fixed portion, or entirely within a single independent segment.

If a MERGE statement appears in an independent segment, any output procedure referenced by that MERGE statement must be entirely within the fixed portion, or entirely within the same independent segment as that MERGE statement.

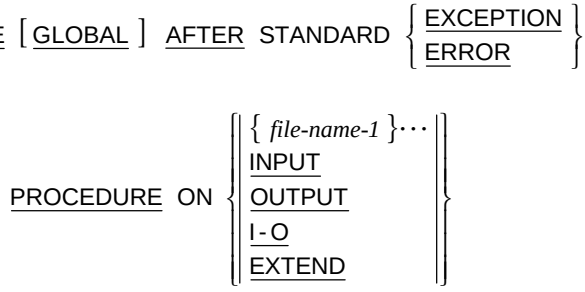
SORT Statement Restrictions

If a SORT statement appears in the fixed portion, any input or output procedures referenced by that SORT statement must be entirely within the fixed portion, or entirely within a single independent segment.

If a SORT statement appears in an independent segment, any input or output procedures referenced by that SORT statement must be entirely within the fixed portion, or entirely within the same independent segment as that SORT statement.

USE Statement

The USE statement specifies procedures for input-output error handling beyond the standard procedures provided by the runtime system. It is a compiler directing statement required in each declarative section.



A USE statement must immediately follow a section header in the declaratives portion of the Procedure Division and must be followed by a separator period. The remainder of the section must consist of zero or more paragraphs that define the procedures to be used.

The USE statement itself is not executed; it defines the conditions calling for the execution of the USE procedure.

A file-name may not be listed in more than one USE statement, nor may it appear more than once in the list of any USE statement. File-names that appear in a USE statement list may not be SORT or MERGE files.

The appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

The INPUT, OUTPUT, I-O and EXTEND phrases may each be specified only once in the declaratives portion of a given Procedure Division.

The words ERROR and EXCEPTION are synonymous in this context.

Declarative procedures may be included in any source program irrespective of whether the program contains or is contained within another program. A declarative is invoked when any of the conditions described in the USE statement which prefaces the declarative occurs while the program is being executed. Only a declarative within the separately compiled program that contains the statement which caused the qualifying condition is invoked when any of the conditions described in the USE statement which prefaces the declarative occurs while that separately compiled program is being executed. If no qualifying declarative exists in the separately compiled program, no declarative is executed.

During the execution of an input-output statement, the runtime system executes the section associated with a USE statement under these conditions:

- An invalid key condition occurs and there is no INVALID KEY phrase in the input-output statement.
- An at end condition occurs and there is no AT END phrase in the input-output statement.
- Some other exception or error condition arises.

The USE section is executed as if it were the operand of a Format 1 PERFORM statement, after having stored the I-O status value into the associated file status data item if there is one.

In circumstances where it is appropriate to do so, the system standard input-output error recovery procedures are also performed.

The rules that determine which USE procedure is to be executed are as follows:

1. If *file-name-1* is specified in the USE statement, the associated procedure is executed when the situation defined above arises during the execution of an input-output statement that refers to *file-name-1*.
2. If the INPUT phrase is specified in the USE statement, the associated procedure is executed when the situation defined above arises during the execution of an input-output statement that refers to any file that is open in the input mode or is in the process of being opened in the input mode, provided the file is not referenced explicitly by name in another USE statement.
3. If the OUTPUT phrase is specified in the USE statement, the associated procedure is executed when the situation defined above arises during the execution of an input-output statement that refers to any file that is open in the output mode or is in the process of being opened in the output mode, provided the file is not referenced explicitly by name in another USE statement.
4. If the I-O phrase is specified in the USE statement, the associated procedure is executed when the situation defined above arises during the execution of an input-output statement that refers to any file that is open in the I-O mode or is in the process of being opened in the I-O mode, provided the file is not referenced explicitly by name in another USE statement.
5. If the EXTEND phrase is specified in the USE statement, the associated procedure is executed when the situation defined above arises during the execution of an input-output statement that refers to any file that is open in the extend mode or is

in the process of being opened in the extend mode, provided the file is not referenced explicitly by name in another USE statement.

When the execution of the USE procedure is complete, control returns to the runtime system. The runtime system then resumes execution of the COBOL program at the next executable statement following the input-output statement whose execution caused the exception or error.

When there is no applicable USE procedure and a critical error occurs for an input-output statement, the runtime system produces an error message and terminates execution of the run unit. This behavior can be configured to allow the program to continue as if a default empty USE procedure were applicable. See the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide* for, information on configuring this behavior.

Within a USE procedure there must be no reference to any nondeclarative procedure. 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 the procedures associated with a USE statement.

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.

Special precedence rules are followed when programs are contained within other programs. In applying these rules, only the first qualifying declarative will be selected for execution. The declarative which is 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 which was last examined for a qualifying declarative.

3. Any declarative selected by applying rule 2 to each more inclusive containing program until rule 2 is applied to the outermost program. If no qualifying declarative is found, none is executed.

USE Statement Example

```
PROCEDURE DIVISION.
DECLARATIVES.
I-O-ERROR SECTION.
    USE AFTER STANDARD EXCEPTION PROCEDURE ON I-O.
I-O-ERROR-ROUTINE.
    DISPLAY "Error for file in I-O open mode.".
    ACCEPT CONTINUE-FLAG POSITION 0 PROMPT.
    IF CONTINUE-FLAG = "NO" STOP RUN.
END DECLARATIVES.
```

Common Rules

Subscript Evaluation

Unless otherwise specified by the rules for a specific statement, any subscripts that appear in an individual statement are evaluated only once as the first operation of the execution of that statement.

Arithmetic Statements

The arithmetic statements are ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT. They have several features in common that are discussed in this section.

Modes of Operation

The data descriptions of the operands in an arithmetic statement need not be the same; any necessary mode conversion and decimal point alignment is supplied throughout the calculation.

Arithmetic operations are done in binary, packed decimal or unpacked decimal mode depending on the operation and on the usage of the operands. If the operation is division or exponentiation it is done in unpacked decimal mode, first converting the values of one or both operands to that mode as necessary. When both operands of an addition or subtraction operation are binary, and they do not have the same number of positions to the right of the decimal point, the operation is done in unpacked decimal mode, first converting the values of both operands to that mode. Other operations are done in the higher mode of the two operands, with binary being treated as the lowest mode and unpacked decimal the highest. If the two operands are of the same mode the operation is

done in that common mode; otherwise, the value of the operand having the lower mode is converted to the higher mode, and the operation is done using the converted value.

Composite Size

The composite size of specified operands in an arithmetic statement other than COMPUTE must not be greater than 30 digits. The specified operands in an ADD or SUBTRACT statement are those operands that contribute values to the final result; operands that serve only as receiving operands are not contributing operands. For example, in the statement ADD A B GIVING C, A and B are contributing operands, but C is not. In the statement ADD P TO Q, both P and Q are contributing operands. In the statement SUBTRACT X FROM Y GIVING Z, X and Y are contributing operands but Z is not.

The specified operands in a MULTIPLY or DIVIDE statement are all the receiving operands except for the operand of the REMAINDER phrase.

The composite size of a set of operands is the size that results when the operands are aligned on their decimal points and the maximum number of positions to the left of the common decimal point position is added to the maximum number of positions to the right of the common decimal point position. For example, if A is defined as PIC 9(8)V9(4) and B is defined as PIC 9(3)V9(6), the composite size of A and B is 8 + 6 = 14. The “phantom” positions resulting from the use of P in the PICTURE character-string are counted in determining the composite size. For example, if X is defined as PIC P(8)9(6) and Y is defined as PIC 9(8)P(10), the composite size of X and Y is 14 + 18 = 32, which exceeds the limit of 30.

ROUNDED Phrase

If, after decimal point alignment, the number of places in the fractional part of the result of an arithmetic operation is greater than the number of places provided for the fractional part of the resultant identifier, truncation is relative to the size provided for the resultant identifier. When the ROUNDED phrase is specified in the arithmetic statement, the absolute value of the resultant identifier is increased by one in the low-order digit position 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 symbol P in the PICTURE character-string for that resultant identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

Size Error Condition

The size error condition occurs under any of the following circumstances:

- Violation of the rules for evaluation of exponentiation always terminates the arithmetic operation and always causes a size error condition.
- Division by zero always terminates the arithmetic operation and always causes a size error condition.
- If, after decimal point alignment and rounding (if specified), the absolute value of a result exceeds the largest value that can be contained in a resultant identifier, a size error condition exists. If the usage of a resultant identifier is binary, the largest value that can be contained in it is the maximum value implied by its PICTURE character-string. However, if a binary allocation override was specified that forced allocation of fewer bytes than needed to support the maximum value implied by its PICTURE character-string, then the maximum value is determined by the maximum value supported by the number of bytes specified in the binary allocation override and, for signed numbers the maximum for the absolute value of negative values is one greater than the maximum for positive numbers. For example, a one-byte signed binary data item can contain the values -128 to +127; the size error condition will exist on an attempt to store a value less than -128 or greater than +127 into a an item described as PIC S9(3) BINARY(1).

If the SIZE ERROR phrase is specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement:

- The values of resultant identifiers for which a size error condition exists remain unchanged from the values they had before execution of the arithmetic statement.
- The values of resultant identifiers for which no size error condition exists are the same as they would have been if the size error condition had not resulted for any of the resultant identifiers.
- After completion of the arithmetic operations, control is transferred to *imperative-statement-1* in the SIZE ERROR phrase and execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of execution of *imperative-statement-1*, control is transferred to the end of the arithmetic statement and the NOT SIZE ERROR phrase, if specified, is ignored.

If the SIZE ERROR phrase is not specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement:

- The values of resultant identifiers for which a size error condition exists are undefined.
- The values of resultant identifiers for which no size error condition exists are the same as they would have been if the size error condition had not resulted for any of the resultant identifiers.
- After completion of the arithmetic operations, control is transferred to the end of the arithmetic statement and the NOT SIZE ERROR phrase, if specified, is ignored.

If the size error condition does not exist after the execution of the arithmetic operations specified by an arithmetic statement, the SIZE ERROR phrase, if specified, is ignored and control is transferred to the end of the arithmetic statement or to *imperative-statement-2* in the NOT SIZE ERROR phrase if it is specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of execution of *imperative-statement-2*, control is transferred to the end of the arithmetic statement.

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, *imperative-statement-1* in the SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.

Overlapping Operands

When a sending and a receiving data item in any statement share a part or all of their storage areas, yet are not defined by the same data description entry, the result of the execution of such a statement is undefined. For statements in which the sending and receiving data items are defined by the same data description entry, the results of the execution of the statement may be defined or undefined depending on the general rules associated with the applicable statement. If there are no specific rules addressing such overlapping operands, the results are undefined.

In the case of reference modification, the unique data item produced by reference modification is not considered to be the same data description entry as any other data description entry. Therefore, if an overlapping situation exists, the results of the operation are undefined.

Incompatible Data

During the execution of the object program, the actual content of a data item is presumed to agree with the class of the data item as specified by its PICTURE clause. No checking is done by the runtime system to detect violations of this requirement, and results are undefined when violations occur. It is particularly important to ensure that the content of a data item described as numeric is in fact numeric when it is used in an arithmetic context.

This rule is suspended for a data item used as the operand of a class condition. Thus, in circumstances in which it is necessary to refer to a data item in an arithmetic context, and it is not certain that the content of the data item is compatible with that type of reference, an IF NUMERIC test should be applied.

Arithmetic Expressions

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operators and parentheses are given in Table 5-1.

Table 5-1: Combination of Symbols in Arithmetic Expressions

First Symbol	Second Symbol				
	Operand	*** / – + ****	**Unary + or –**	**(**	**)**
Operand (an identifier or literal)	☐	☐	■	☐	☐
* / + – **	■	☐	■	■	☐
Unary + or –	■	☐	☐	■	☐
(	■	☐	■	■	☐
)	☐	■	☐	☐	■

■ *A permissible pair of symbols.*

☐ *An invalid pair of symbols.*

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

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 must be preceded by a space and followed by a space. See Table 5-2.

Table 5-2: Arithmetic Operators

Type	Operator	Meaning
BINARY	+	Addition.
	–	Subtraction.
	*	Multiplication.
	/	Division.
	**	Exponentiation.
UNARY	+	The effect of multiplication by the numeric literal +1.
	–	The effect of multiplication by the numeric literal –1.

Formation and Evaluation Rules

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:

1. Unary plus and minus
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction

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.

The ways in which operands, operators and parentheses may be combined in an arithmetic expression are summarized in Table 5-1 on page 5-21.

An arithmetic expression may begin only with one of the following symbols:
(+ – or an operand. An arithmetic expression may end only with a) or an operand. There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis.

The following rules apply to evaluation of exponentiation in the following arithmetic expression:

arithmetic-expression-1 ** **arithmetic-expression-2**

arithmetic-expression-1 provides the base value and *arithmetic-expression-2* provides the exponent value.

1. If the value of the base is zero, the exponent value must be greater than zero; otherwise, the size error condition exists.

2. If the value of the base is negative, the exponent value must be an integer; otherwise, the size error condition exists.

Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands, receiving data items, or both.

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 may be used in the EVALUATE, IF, PERFORM, and SEARCH statements. There are two categories of conditions associated with conditional expressions: simple conditions and complex conditions. Each may be enclosed within any number of paired parentheses, in which case its category is not changed.

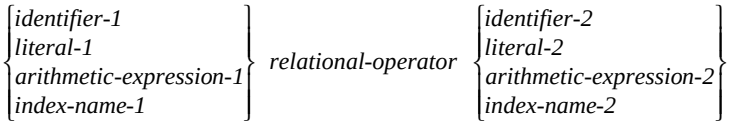
Simple Conditions

The simple conditions are relation, class, sign, condition-name and switch-status. A simple condition has a truth value of true or false. A simple condition enclosed in parentheses has the same truth value as the simple condition standing alone.

Relation Condition

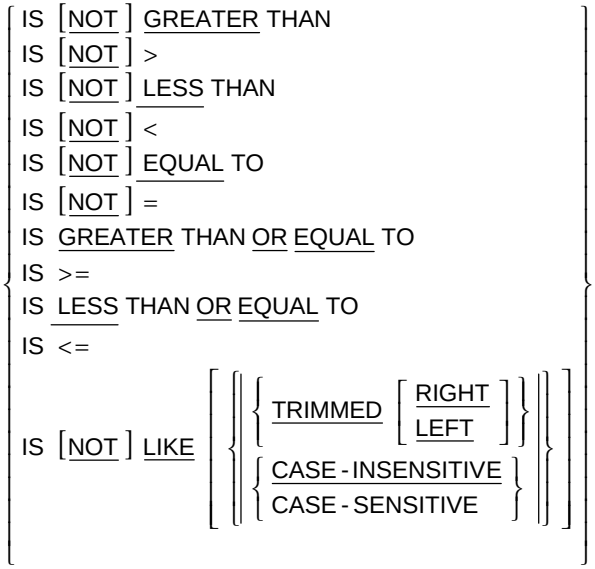
A relation condition causes a comparison of two operands, each of which may be the data item referenced by an identifier, a literal, an arithmetic expression or an index-name. A relation condition has the truth value of true if the relation exists between the operands; otherwise, the relation condition has the truth value of false.

The general format of a relation condition is:



The operand to the left of the *relational-operator* is called the subject of the condition; the operand to the right is called the object of the condition, or, in the case of the LIKE relational operator, the pattern of the condition (see page 5-27).

The general format for the *relational-operator* is:



The relational operator specifies the type of comparison to be made in a relation condition. A space must precede and follow each reserved word that makes up the relational operator. When used, NOT and the next keyword 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 than comparison. The relational operator IS NOT GREATER THAN is equivalent to IS LESS THAN OR EQUAL TO, and the relational operator IS NOT LESS THAN is equivalent to IS GREATER THAN OR EQUAL TO.

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. POINTER usage in RM/COBOL is not a numeric usage; pointer operands may only be compared to other pointer operands, which include the figurative constant NULL (NULLS) and the ADDRESS special register.

The meanings of the relational operators are given in Table 5-3.

Table 5-3: Relational Operators

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than.
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than.
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to.
IS GREATER THAN OR EQUAL TO IS >=	Greater than or equal to.
IS LESS THAN OR EQUAL TO IS <=	Less than or equal to.
IS [NOT] LIKE	Pattern matches or not matches subject.

Note The required relational characters >, < and = are not underlined to avoid confusion with other symbols such as ≥ (greater than or equal to).

Comparison of Numeric Operands

For operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands, aligned by their decimal points. The lengths of the operands, in terms of number of digits represented, are not significant. Zero is considered a unique value regardless of the sign.

Comparison of numeric operands is permitted regardless of their usage. 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. When a numeric operand is compared with a nonnumeric operand, the following rules apply:

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), and the contents of this alphanumeric data item were then compared to the nonnumeric operand.

2. If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this group item were then compared to the 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 only when the numeric operand is an integer and its usage is DISPLAY.

There are two cases to consider: operands of equal size and operands of unequal size.

Operands of equal size:

- 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 corresponding characters are equal.

- 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.

Operands of unequal size:

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

Comparisons of Index-Names and Index Data Items

If two index-names are compared, the result is the same as if the corresponding occurrence numbers were compared.

For an index-name and a data item (other than an index data item) or literal, the comparison is made between the occurrence number that corresponds to the value of the index-name and the data item or literal.

When a comparison is made between an index data item and an index-name or another index data item, the actual values are compared without conversion to the occurrence number.

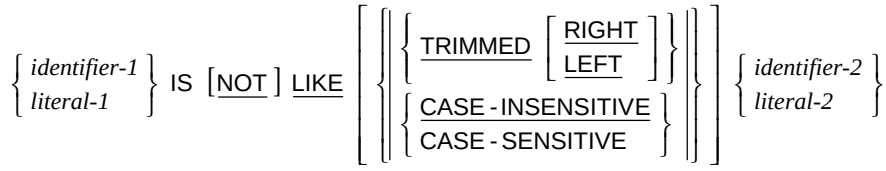
Comparison of an index data item with any data item or literal not specified above is not permitted.

Comparison of Pointer Data Items

For operands that are pointers, a comparison is made with respect to the effective address of the operands. The effective address of a pointer is the sum of the address and offset values for the pointer. A null pointer value (for example, the figurative constant NULL) has an effective address of zero. Thus, a pointer data item is always either equal to or greater than a null pointer value.

LIKE Condition (Special Case of Relation Condition)

The general format for the LIKE condition is:



identifier-1 must refer to an alphanumeric data item.

literal-1 and *literal-2* must be nonnumeric literals.

identifier-2 must refer to an alphanumeric data item or a pointer data item.

The data item referenced by *identifier-1* or the value of *literal-1* is the subject of the condition.

The data item referenced by *identifier-2* or the value of *literal-2* is the pattern of the condition. If *identifier-2* refers to an alphanumeric data item, the value of that data item

specifies the pattern as a regular expression. If *identifier-2* refers to a pointer data item, then the value of that data item points to a compiled pattern.

The LIKE condition returns true if the subject value of the condition matches the pattern value of the condition and false otherwise.

Unless otherwise specified by use of the TRIMMED phrase, the entire contents of the subject value must match the pattern value. If the TRIMMED LEFT phrase is specified, leading spaces are ignored. If the TRIMMED RIGHT phrase is specified, trailing spaces are ignored. If the TRIMMED phrase is specified without either the LEFT or RIGHT modifiers, leading and trailing spaces are ignored. The TRIMMED phrase must not be used if the subject data may contain significant spaces that would be ignored as a result of its specification; reference modification of the subject may be necessary to select the significant portion of the data to be matched in this case.

Case is significant for the LIKE condition if the CASE-SENSITIVE phrase is specified or implied, that is, a case-sensitive match of the subject value to the pattern value is done. Case is not significant for the LIKE condition if the CASE-INSENSITIVE phrase is specified, that is, a case-insensitive match of the subject value to the pattern value is done.

The pattern may be specified as a literal, an alphanumeric data item, or a pointer data item, with the following interpretations:

- Literal pattern. The RM/COBOL compiler automatically compiles the pattern specified as *literal-2* during source program compilation. Errors in the pattern, if any, are reported in the compilation listing, including an indication of where in the pattern the problem occurred.
- Alphanumeric data item pattern. The RM/COBOL compiler generates code to compile at runtime a pattern specified as *identifier-2*, where *identifier-2* refers to an alphanumeric data item that contains the pattern. If the data item contains leading or trailing spaces that are not part of the pattern value, reference modification must be used to exclude the spaces. If the pattern contains an error, the LIKE condition will return a false (non-matching) result without any indication that an error occurred. The pattern will be re-compiled each time the condition is executed, regardless of whether the pattern value has changed.
- Pointer data item pattern. If a pattern must be variable at runtime, but is used multiple times for a given pattern value, compiling the pattern once and specifying a pointer to the compiled result can enhance performance. In this case, the data item referenced by *identifier-2* must be a pointer data item, the value of which has been previously set by using the subprogram library routine, C$CompilePattern, as described in Appendix F, *Subprogram Library*, of the *RM/COBOL User's Guide*. When called, this routine indicates if the pattern contains an error and also provides an easy method of stripping trailing spaces in the pattern value. Therefore, this method is preferable to using an alphanumeric data item directly in the LIKE

condition regardless of performance issues. The LIKE condition returns a false (non-matching) result if the pattern is specified as a null valued pointer or if the pointer does not point to a compiled pattern.

A pattern is specified by a regular expression. A regular expression is a string that uses expressions similar to arithmetic expressions to specify the rules for matching. Various operators are used to combine smaller expressions. The formal grammar for regular expressions is given on page 5-36. The regular expressions used in the LIKE condition are the same as those specified for XML (eXtensible Markup Language) schema. A regular expression is composed as follows:

1. Any character other than the special characters specified in item 2 are ordinary characters. An ordinary character is a one-character regular expression that matches itself. For example, "A" matches the string "A" and "3" matches the string "3". Characters may be specified using XML character references as "&#*d*;", where *d* is one or more decimal digits that provide the decimal representation of the Unicode code-point for the character, or as "&#x*h*;", where *h* is one or more hexadecimal digits that provide the hexadecimal representation of the Unicode code-point for the character. Also, the recognized XML entity references are illustrated in Table 5-4.

Table 5-4: XML Entity References

Entity Reference	Character
&	&
'	'
<	<
>	>
"	"

Recognition and conversion of XML character references and XML entity references occur before a character is interpreted within the regular expression. Incomplete sequences are treated as the literal sequence of characters. For example, "&amp", which is missing the required semicolon, represents the character sequence '&', 'a', 'm' and 'p'. Such incomplete sequences do not cause an error because of the incompleteness, but may cause an error if the literal sequence is not valid in the context in which it appears. For example, "\." is equivalent to "\.", which is a valid escaped period, but "\&#x2E" includes the sequence "\&", which is not a valid escape sequence and would, therefore, cause an error. No part of the XML character reference or XML entity reference sequence may be represented using an XML character reference or XML entity reference. For example, the sequence "&" is recognized as literally "&" and is not further converted to "&".

2. The characters “.” (period), “\” (back slash), “*” (asterisk), “+” (plus sign), “?” (question mark), “|” (vertical bar), “(” (left parenthesis), “)” right parenthesis, “[” (left bracket), “]” (right bracket), “{” (left brace), and “}” right brace are special characters that act as operators, which are explained individually in the items that follow.

3. The special character “.” (period) matches any character other than newline (0Ah) or return (0Dh). For example, "." matches any of the strings "A" or "B" or "9".

4. The special character “\” (backslash) begins an escape sequence. Escape sequences may be single-character escapes (see the following table), multi-character escapes (see page 5-31), or category escapes (see page 5-32).

   Single-character escapes match a single character and exist because that character is usually difficult or impossible to write directly into a regular expression. The valid single-character escapes are shown in Table 5-5.

Table 5-5: Regular Expression Single-Character Escape Sequences

Escape Sequence	Character
\n	newline (
)
\r	return (
)
\t	horizontal tab ( )
\\	\
|	|
\.	.
\-	-
\^	^
\?	?
*	*
\+	+
\{	{
\}	}
\(	(
\)	)
\[	[
\]	]

Multi-character escapes match commonly used sets of characters without having to write a character class expression to describe the set of characters to be matched. Table 5-6 lists and describes the valid multi-character escapes.

Table 5-6: Regular Expression Multi-Character Escape Sequences

Escape Sequence	Equivalent Character Class	Meaning
.	[^\n\r]	Any character except newline or return.
\s	[\t\n\r]	White space.
\S	[^\s]	Not whitespace.
\i	[\p{L}_:]	Initial name characters (of XML).
\I	[^\i]	Not initial name characters (of XML).
\c	[\i\d\.·-]	Name characters (of XML). (See Note 2 below.)
\C	[^\c]	Not name characters (of XML).
\d	\p{Nd}	Numeric digits.
\D	[^\d]	Not numeric digits.
\w	[&#x00;-ÿ-[\p{P}\p{Z}\p{S}\p{C}]]	All characters except punctuation, separator, symbol and other characters. (See Note 1 below and property definitions in Table 5-7.)
\W	[^\w]	Punctuation, separator, symbol and other characters. (See Note 1 below.)

Note 1 The definitions of the "\w" and "\W" sequences are subject to change, so these sequences should be avoided until they are clarified. The XML schema definition of "\w" is unclear because it is described as "all characters except the set of 'punctuation', 'separator', and 'control' characters". This informal description differs from its formal definition of [&#x00;-&#x10FFFF;]-[\p{P}\p{S}\p{C}], which is all characters except the set of punctuation, symbol and other characters. This could mean that the formal definition should be [&#x0000;-&#x10FFFF;]-[\p{P}\p{Z}\p{C}], [&#x00;-&#x10FFFF;]-[\p{P}\p{Z}\p{Cc}] or [&#x00;-&#x10FFFF;]-[\p{P}\p{Z}\p{S}\p{C}]. Since "\w" probably stands for "the word class of characters", the latter may be the correct interpretation and is the one currently implemented in RM/COBOL. In the regular expressions of the Perl language, "\w" matches alphanumeric characters including "_", which strictly

interpreted would be [\p{L}\p{N}_]. Unicode classifies "_" in the "Pc" category, so excluding punctuation characters excludes the "_" character. The definition of "\W", the characters not in "\w", depends on the definition of "\w" and is, therefore, similarly unclear.

Note 2 The B7h code point in Unicode is the "MIDDLE DOT" extender character and is classified as a name character. Therefore, XML name characters include this code point value.

Category escapes match sets of characters based on their Unicode category. The set of characters that have Unicode property *X* is designated with "\p{*X*}". The complement of this set, that is, all characters that do not have Unicode property *X*, is specified as "P{*X*}". Unicode property designators are an uppercase letter optionally followed by a lowercase letter. The valid character property designators from the Unicode standard are shown in Table 5-7.

Table 5-7: Unicode Valid Character Property Designators

Category	Property Designator	Character Class
Letters	L	All letters.
	Lu	Uppercase letters.
	Ll	Lowercase letters.
	Lt	Title case letters.
	Lm	Modifier letters.
	Lo	Other letters.
Marks	M	All marks.
	Mn	Non-spacing marks.
	Mc	Spacing combining marks.
	Me	Enclosing marks.
Numbers	N	All numbers.
	Nd	Decimal digit numbers.
	Nl	Letter numbers.
	No	Other numbers.
Punctuation	P	All punctuation.
	Pc	Connector punctuation.
	Pd	Dash punctuation.
	Ps	Open punctuation.
	Pe	Close punctuation.
	Pi	Initial quote punctuation.
	Pf	Final quote punctuation.
	Po	Other punctuation.

Table 5-7: Unicode Valid Character Property Designators (*Cont.*)

Category	Property Designator	Character Class
Separators	Z	All separators.
	Zs	Space separators.
	Zl	Line separators.
	Zp	Paragraph separators.
Symbols	S	All symbols.
	Sm	Math symbols.
	Sc	Currency symbols.
	Sk	Modifier symbols.
	So	Other symbols.
Other	C	All others.
	Cc	Control others.
	Cf	Format others.
	Co	Private use others.
	Cn	Not assigned others.

For example, the pattern value "\p{Nd}" matches any decimal digit character and the pattern value "\P{Nd}" matches any character other than a decimal digit character.

In addition to specifying any of the character property designators above, the character category escape can also specify any of the Unicode character blocks. In this case, the property is specified as Is*BlockName*, where *BlockName* is the Unicode block name with all white space stripped out. Since this implementation only supports 8-bit characters, only the character blocks IsBasicLatin (characters 00h through 7Fh) and IsLatin-1Supplement (characters 80h through FFh) are non-empty. For example, the pattern value "\p{IsBasicLatin}" matches any character in the range 00h through 7Fh, and the pattern value "\P{IsBasicLatin}" matches any character that is not 00h through 7Fh.

5. The special characters, “[” (left bracket) and “]” (right bracket), are used to define a one-character character class regular expression. The character class matches any of the characters specified between the brackets, except that, when the “^” (caret) character is the first character after the left bracket the class matches any character not specified between the brackets. Special characters (listed in item 2), other than “\”, “[”, and “]”, lose their special meaning when contained in brackets (that is, they represent themselves in the character class). A range of characters may be specified with the “-” (hyphen) character separating two other characters. To include a “^” in the character class, include it anywhere except as the first character after the left bracket (if the “^” is the only character in the class, omit the brackets). To include a “-” in the character class, include it as the first (or second if the first character is a

"^") or last character between the brackets or use the escape sequence "\-" to specify the character. To include a "\", "[", or "]" in the character class, use the escape sequences "\\", "\[", or "\]", respectively. For example, "[0-9]" matches a decimal digit character and "[^0-9]" matches any character except a decimal digit character. The second character in a hyphenated character range must not be less than the first character.

6. Within a character class expression, a character class may be subtracted by using the "-" followed by another character class expression. For example, "[\p{P}-[;:]]" defines a character class that includes all the punctuation characters except for semicolon and colon. A character class subtraction must be the last portion of a character class expression before the closing "]" for the containing character class expression, but may contain character class subtractions within itself. When a character class is negated, that is, begins with the "^" character, the negation takes place before the subtraction, that is, the negation has higher precedence than the class subtraction. For example, "[^A-F-[U-Z]]", the characters not in [A-F] less the characters in [U-Z], is equivalent to "[^A-FU-Z]", the characters not in [A-FU-Z].

7. Subexpressions may be concatenated by juxtaposition in left to right order. For example, "AB." matches "ABC" or "ABD" or "AB3" or any other three-character string that begins with "AB". As another example, "A[BC]D" matches "ABD" or "ACD".

8. Two subexpressions may be combined with the "|" infix operator to specify alternatives. If either of the subexpressions matches the current position in the subject string, the regular expression formed in this way matches. For example, "PRE|PER" matches "PRE" or "PER".

9. The special character "*" causes the preceding subexpression to be matched zero or more times. For example, the string "AB*C" matches "AC" or "ABC" or "ABBBBC".

10. The special character "+" causes the preceding subexpression to be matched one or more times. For example, the pattern "$[0-9]+\.00" matches the strings "$0.00" or "$1.00" or "$392.00", but does not match the string "$.00" since there are no digits before the decimal point.

11. The special character "?" causes the preceding subexpression to be matched zero or one times, that is, the preceding subexpression is optional in matching. For example, the string "-$?123" matches the string "-$123" or "-123".

12. The special characters "{" and "}" are used to define repetition of the preceding subexpression by a specified range. "{*n*}" or "{*n*,*n*}" matches exactly *n* occurrences. "{*n*,}" matches *n* or more occurrences. "{*n*,*m*}" matches from *n* to *m* occurrences. *n* and *m* must be decimal integers in the range 0 to 65535, and *n* must be less than or

equal to *m*. When a choice is allowed, the longest matching string in the subject is matched. For example "(A{2})*" matches zero or more pairs of “A” characters in the subject string, "A{3,}" matches three or more successive “A” characters in the subject string, and "A{3,5}" matches from 3 to 5 successive “A” characters in the subject string.

Note “{0}” or “{0,0}” cause the previous subexpression to be ignored. “{1}” or “{1,1}” are redundant since they are equivalent to the default. “{0,}” is equivalent to “*”, “{1,}” is equivalent to “+”, and “{0,1}” is equivalent to “?”.

13. The order of precedence for operators from highest to lowest is escape (with “\”), class definition (with “[” and “]”), repetition (with “*”, “+”, “?”, or “{}”), concatenation, and alternation (with “|”). The order of precedence for repetition, concatenation, and alternation can be overridden by use of parentheses. For example:

    - "AB|CD" matches "AB" or "CD", because concatenation has higher precedence than alternation;
    - "A(B|C)D" matches "ABD" or "ACD", because the parentheses override the precedence order;
    - "ABC*" matches "ABCCCC", because repetition has higher precedence than concatenation; and
    - "(ABC)*" matches zero or more occurrences of "ABC" in the subject string, because parentheses override the precedence order.

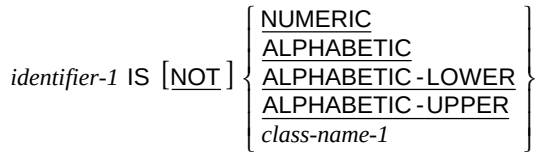
When the TRIMMED phrase is not specified in the LIKE condition, matching is done on the entire contents of the subject value. In this case, the pattern must specify whether trailing spaces are to be included in the match. If the pattern does not allow for trailing spaces and the subject value contains trailing spaces, the LIKE condition result will be false (non-matching). To allow for trailing spaces, the pattern should end with " *", that is, a space followed by the “*” repetition operator. This is not necessary if, for example, the pattern ends with ".*", that is, a period followed by the “*” repetition operator, since this allows any number of any trailing character, including trailing spaces.

Regular expression grammar summary:

```
 [1] regExp          ::= branch ( '|' branch )*
 [2] branch          ::= piece*
 [3] piece           ::= atom quantifier?
 [4] quantifier      ::= [?*+] | ( '{' quantity '}' )
 [5] quantity        ::= quantRange | quantMin | QuantExact
 [6] quantRange      ::= QuantExact ',' QuantExact
 [7] quantMin        ::= QuantExact ','
 [8] QuantExact      ::= [0-9]+
 [9] atom            ::= Char | charClass | ( '(' regExp ')' )
[10] Char            ::= [^.\?*+()|#x5B#x5D]
[11] charClass       ::= charClassEsc | charClassExpr
[12] charClassExpr   ::= '[' charGroup ']'
[13] charGroup       ::= posCharGroup | negCharGroup |charClassSub
[14] posCharGroup    ::= ( charRange | charClassEsc )+
[15] negCharGroup    ::= '^' posCharGroup
[16] charClassSub    ::= ( posCharGroupND | negCharGroupND )
                           '-' charClassExpr
[17] negCharGroupND  ::= '^' posCharGroupND
[18] posCharGroupND  ::= ( XmlCharRef | XmlChar | charClassEsc )+
[19] XmlCharRef      ::= ( '&#' [0-9]+ ';' ) |
                           ( '&#x' [0-9a-fA-F]+ ';' )
[20] XmlChar         ::= [^\#x2D#x5B#x5D]
[21] charRange       ::= seRange | XmlCharRef | XmlCharIncDash
[22] seRange         ::= charOrEsc '-' charOrEsc
[23] charOrEsc       ::= XmlChar | SingleCharEsc
[24] XmlCharIncDash  ::= [^\#x5B#x5D]
[25] charClassEsc    ::= ( SingleCharEsc | MultiCharEsc |
                           catEsc | complEsc )
[26] SingleCharEsc   ::= '\' [nrt\|.?*+(){}#x2D#x5B#x5D#x5E]
[27] catEsc          ::= '\p{' charProp '}'
[28] complEsc        ::= '\P{' charProp '}'
[29] charProp        ::= IsCategory | IsBlock
[30] IsCategory      ::= Letters | Marks | Numbers |
                           Punctuation | Separators |
                           Symbols | Others
[31] Letters         ::= 'L' [ultmo]?
[32] Marks           ::= 'M' [nce]?
[33] Numbers         ::= 'N' [dlo]?
[34] Punctuation     ::= 'P' [cdseifo]?
[35] Separators      ::= 'Z' [slp]?
[36] Symbols         ::= 'S' [mcko]?
[37] Others          ::= 'C' [cfon]?
[38] IsBlock         ::= 'Is' [a-zA-Z [0-9a-zA-Z#x2D]*
[39] MultiCharEsc    ::= '.' | ( '\' [sSiIcCdDwW] )
```

Class Condition

The general format for the class condition is:



The class condition determines whether the current contents of an operand are numeric, alphabetic, alphabetic-lower, alphabetic-upper, or consist only of the characters in the set of characters specified by a CLASS clause defined in the SPECIAL-NAMES paragraph of the Environment Division. The class of an operand is determined as follows:

- An operand is numeric if its contents consist entirely of the characters 0, 1, 2, 3, . . ., 9, with or without an operational sign. The specified usage of the operand and its explicit or implicit SIGN clause are taken into account in determining the validity of the digit and sign representation.

- An operand is alphabetic if its contents consist entirely of any combination of the uppercase letters A, B, C, . . ., Z, the lowercase letters a, b, c, . . ., z, or space. It should be noted that this definition of the alphabetic test is not the same as the definition of the alphabetic test in previous versions of COBOL. In order to achieve compatibility with the earlier versions of COBOL, two courses of action are possible: either change the source program to use the alphabetic-upper test in place of the alphabetic test, or make use of the Compile Command option that causes the RM/COBOL compiler to treat alphabetic tests as if they were alphabetic-upper tests. The *RM/COBOL User's Guide* contains further information on this topic.

- An operand is alphabetic-lower if its contents consist entirely of the lowercase letters a, b, c, . . ., z, or space.

- An operand is alphabetic-upper if its contents consist entirely of the uppercase letters A, B, C, . . ., Z, or space.

- An operand fulfills a class-name test if its contents consist entirely of the characters listed in the definition of *class-name-1* in the SPECIAL-NAMES paragraph.

When used, NOT and the next keyword 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.

In the NUMERIC test, the usage of the operand being tested may be DISPLAY, COMPUTATIONAL, COMPUTATIONAL-3 or COMPUTATIONAL-6.

If the usage of the operand being tested is DISPLAY, then:

1. If the data description of the item being tested indicates 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. The valid operational signs for numeric DISPLAY data items are defined in the discussion of the SIGN clause that begins on page 4-48 and in the discussion of the USAGE clause on page 4-52.

2. 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 usage of the operand being tested is COMPUTATIONAL, the item being tested is determined to be numeric only if each character position contains an unpacked decimal digit, except that, if the data description of the item being tested indicates the presence of an operational sign, the rightmost character position must contain a valid sign. The representation for a negative sign is hexadecimal D. Depending on configured sign representation, the representation for a positive sign may be hexadecimal C, B, or F.

If the usage of the operand being tested is COMPUTATIONAL-3, the item being tested is determined to be numeric only if each character position, except the rightmost, contains two packed decimal digits. The rightmost character position must contain a packed decimal digit in the high order half-byte and a valid sign in the low order half-byte. The representation for a negative sign is hexadecimal D. Depending on configured sign representation, the representation for a positive sign may be hexadecimal C, B, or F.

If the usage of the operand being tested is COMPUTATIONAL-6, the item being tested is determined to be numeric only if each character position contains two packed decimal digits.

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 alphabetic characters and the space.

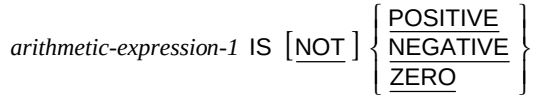
The ALPHABETIC-LOWER test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic-lower only if its contents consist of any combination of the lowercase alphabetic characters a through z and space.

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

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

Sign Condition

The sign condition determines whether the algebraic value of an arithmetic expression is less than, greater than, or equal to zero. The general format for a sign condition is:



When used, NOT and the next keyword 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 value. A value is positive only if it is greater than zero. A value is negative only if it is less than zero. The value zero is neither positive nor negative.

Condition-Name Condition (Conditional Variable)

In a condition-name condition, a conditional variable is tested to determine whether its value is equal to one of the values associated with a condition-name. The general format for the condition-name condition is:

condition-name-1

If *condition-name-1* is associated with a range of values, the conditional variable is tested to determine if its value falls within 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.

Switch-Status Condition

A switch-status condition determines the on or off status of a software switch. The switch-name 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:

condition-name-2

The result of the test is true if the switch is set to the specified position corresponding to *condition-name-2*.

Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions and complex conditions with logical connectors (logical operators AND and OR) or by negating these conditions with logical negation (the logical operator NOT). The truth value of a complex condition, whether parenthesized or not, is the truth value that results from the interaction of the stated logical operators on the individual truth values of the constituent simple conditions.

The logical operators and their meanings are shown in Table 5-8.

Table 5-8: Logical Operators

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 Conditions

A condition is negated by the use of the logical operator NOT which reverses the truth value of the condition to which it is applied. Thus the truth value of a negated condition is true only if the truth value of the condition is false; the truth value of a negated condition is false only if the truth value of the condition is true. The inclusion in parentheses of a negated condition does not change the truth value.

The general format for a negated condition is:

NOT *condition-1*

Combined Conditions

A combined condition results from connecting conditions with one of the logical operators AND or OR. The general format of a combined condition is:

condition-2 { { AND / OR } *condition-3* }...

condition-2 and *condition-3* may be one of the following:

- Simple condition.
- Negated condition.
- Combined condition.
- Negated combined condition; that is, the NOT logical operator followed by a combined condition enclosed within parentheses.
- Combinations of the above.

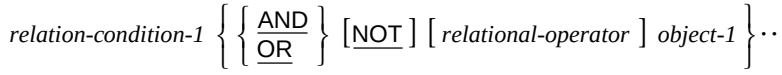
Although parentheses need never be used when AND or OR (but not both) is used exclusively in a combined condition, parentheses may be used to affect the final truth value when a mixture of AND, OR and NOT is used.

Abbreviated Combined Relation Conditions

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

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

The format for an abbreviated combined relation condition is:



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

- If the word immediately following NOT is GREATER, >, LESS, <, EQUAL or =, the NOT participates as part of the relational operator.
- In all other circumstances, the NOT is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

Condition Evaluation Rules

Condition evaluation rules indicate 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 such that each left parenthesis is to the left of its corresponding right parenthesis.

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:

- Truth values for simple conditions are established.
- Truth values for negated simple conditions are established.
- Truth values for combined conditions are established: AND logical operators followed by OR logical operators.
- Truth values for negated combined conditions are established.
- 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.

Sequential Organization Input-Output

The sequential organization input-output statements in the Procedure Division are the CLOSE, DELETE FILE, OPEN, READ, REWRITE, UNLOCK and WRITE statements.

Function

Sequential organization input-output provides a capability to access records of a file in an established sequence. The sequence is established as a result of writing the records to the file.

Organization

Sequential files are organized such 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, these relationships do not change except when records are added to the end of the file.

Access Mode

Only the sequential access mode is available for files whose organization is sequential. In the sequential access mode, the sequence in which records are accessed is the order in which the records were originally written.

File Position Indicator

The file position indicator is a concept used to facilitate specification of the next record to be accessed within a given file during certain sequences of input-output operations. The concept of the file position indicator has no meaning for a file opened in the output or extend mode. The setting of the file position indicator is affected only by the CLOSE, OPEN and READ statements.

I-O Status

If the FILE STATUS clause is included in a file control entry, it defines a two-character file status data item for that file. During the execution of each input-output statement that refers to such a file, the runtime system stores a value into the file status data item. Storage of the value is done before the execution of any associated imperative statement and before any applicable USE procedure is executed. The value can be used by the program to determine the status of that input-output operation. The value that is stored into the file status data item is called the I-O status value.

The I-O status value indicates the status of an input-output operation. It also determines whether an applicable USE procedure should be executed: if one of the conditions listed under the heading “Successful Completion” results, an applicable USE procedure is not executed; if any other condition results, such a procedure may be executed depending on rules stated in the section of this chapter that discusses the USE statement (see page 5-13).

Certain classes of I-O status values indicate critical error conditions. They are the ones that begin with the digits 3, 4 and 9. When such conditions arise, certain system-standard error correction procedures may be tried first, depending on the nature of the problem. If

they are not successful in clearing the problem, either a user-specified USE procedure is executed (if one is applicable) and execution of the program continues, or a runtime error message is produced and execution of the run unit terminates.

Upon completion of the input-output operation the I-O status value expresses one of the following conditions:

- Successful Completion. The input-output statement was executed successfully and no exceptional conditions arose. The left character of the I-O status value is 0 for these cases.

- At End. A sequential READ statement was not executed successfully because of an at end condition. The left character of the I-O status value is 1 for this case.

- Permanent Error. The input-output statement was not executed successfully because of an error that precludes further processing of the file. The problem could be a violation of an external boundary, or a hardware input-output error such as a data check, parity error, transmission error, and so forth. The left character of the I-O status value is 3 for these cases.

- Logic Error. The input-output statement was not executed successfully because an improper sequence of input-output statements was performed on the file, or because of a violation of a user-defined limit. The left character of the I-O status value is 4 for these cases.

- General Error. The input-output statement was not executed successfully because of a condition that is specified by the right character of the I-O status value. The left character of the I-O status value is 9 for these cases.

It should be noted that the I-O status values specified here differ in many respects from the ones defined in earlier versions of RM/COBOL. The new values comply with the American National Standard COBOL 1985 whereas the old values comply with ANSI COBOL 1974. In the following list, the old values are shown in square brackets following the new values when the two values are not the same. In situations where it is necessary to preserve compatibility with earlier versions of RM/COBOL in this respect, two courses of action are possible: either modify the text of the source program to use the new set of status values, or make use of the Compile Command option that causes the compiler to treat the entire program as an ANSI COBOL 1974 program. That option and the language features it controls are discussed in detail in the *RM/COBOL User's Guide*.

The following list shows the possible I-O status values that can arise as a result of executing an input-output statement that refers to a sequential file:

- Successful Completion
  - I-O Status Value=00. The input-output statement is successfully executed and no further information is available concerning the operation.
  - I-O Status Value=04 [97]. A READ statement executed successfully but the length of the record being processed does not conform to the fixed file attributes for the file.
  - I-O Status Value=05. The input-output statement is successfully executed but the file is not present at the time the input-output statement is executed.
    - For a DELETE FILE statement, the referenced file is not available.
    - For an OPEN statement, the referenced optional file is not present. If the open mode is I-O or extend, the file has been created.
  - I-O Status Value=07. The input-output statement executed successfully. However, for a CLOSE statement with the NO REWIND, REEL/UNIT, or FOR REMOVAL phrase or for an OPEN statement with the NO REWIND phrase, the referenced file is on a non-reel/unit medium.
- At End Condition with Unsuccessful Completion

  I-O Status Value=10. A sequential READ statement is attempted and no next logical record exists in the file because the end of the file has been reached, or a sequential READ statement is attempted for the first time on an optional input file that is not present.
- Permanent Error Condition with Unsuccessful Completion
  - I-O Status Value=30. A permanent error exists and no further information is available concerning the input-output operation.
  - I-O Status Value=34. A permanent error exists because of an attempt to write beyond the externally defined boundaries of a sequential file.
  - I-O Status Value=35 [94]. A permanent error exists because an OPEN statement with the INPUT, I-O, or EXTEND phrase is attempted on a nonoptional file that is not present.
  - I-O Status Value=37 [90, 95]. A permanent error exists because an OPEN statement is attempted on a file that does not support the open mode specified in

the OPEN statement, or a DELETE FILE statement refers to a protected file. For OPEN statements, the possible violations are as follows:

- 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 that are permitted for a sequential file when opened in the I-O mode.
- The INPUT phrase is specified but the file does not support read operations.

– I-O Status Value=38 [93]. A permanent error exists because an OPEN or DELETE FILE statement is attempted on a file previously closed with lock.

– I-O Status Value=39 [94]. An OPEN or DELETE FILE statement is unsuccessful because of an incompatibility between the fixed file attributes and the attributes specified for the file in the program.

- Logic Error Condition with Unsuccessful Completion

  – I-O Status Value=41 [92]. An OPEN statement is attempted for a file that is already open, or a DELETE FILE statement is attempted for an open file.

  – I-O Status Value=42 [91]. A CLOSE statement is attempted for a file that is not open.

  – I-O Status Value=43 [90]. A REWRITE statement is attempted for a mass storage file, and the last input-output statement executed for the file was not a successfully executed READ statement.

  – I-O Status Value=44 [97]. A boundary violation exists either because of an attempt to write or rewrite a record whose length is longer or shorter than the limits established by the RECORD IS VARYING clause, or because of an attempt to rewrite a record that is not the same size as the record being replaced.

  – I-O Status Value=46 [96]. A sequential READ statement is attempted on a file open in the input or I-O mode and no valid next record has been established either because the preceding READ statement caused an at end condition, or because the preceding READ statement was unsuccessful for some other reason.

  – I-O Status Value=47 [90, 91]. A READ statement is attempted on a file not open in the input or I-O mode.

- I-O Status Value=48 [90, 91]. A WRITE statement is attempted on a file not open in the output or extend mode.

- I-O Status Value=49 [90, 91]. A REWRITE statement is attempted on a file not open in the I-O mode.

- General Error

  - I-O Status Value=93. An OPEN statement is attempted on a file that is not available. The availability of a file is determined by several factors, including the lock mode. See the *RM/COBOL User's Guide* for details on the availability of a file.

  - I-O Status Value=94. An OPEN statement is attempted at a time when there is insufficient available memory to provide the required supplementary input-output areas and control structures, or an OPEN statement is attempted for a file that has an attribute that is not supported, or an OPEN statement is attempted for a file that has file attributes that are inconsistent among themselves.

  - I-O Status Value=97. A REWRITE or WRITE statement is attempted while the record area contains one or more characters that are not legal for a line sequential file after mapping through the applicable code set.

  - I-O Status Value=98. Defective record structure has been found in the file.

  - I-O Status Value=99. A READ or REWRITE statement is attempted that refers to a record locked by another concurrent user. This I-O status value is returned only when the referenced file has an associated file status data item and there is an applicable USE procedure; when this is not the case, the program waits for the record to become available.

At End Condition

The at end condition can occur as a result of the execution of a Format 1 READ statement. Details regarding the circumstances that cause an at end condition are presented in the discussion of the Format 1 READ statement that begins on page 6-146.

If the at end condition arises, execution of the READ statement is unsuccessful and the positioning of the file is not changed. The NOT AT END phrase and its imperative statement, if present, are ignored, and the following actions occur:

1. If there is a file status data item associated with the file, the appropriate I-O status value (10) is stored into it.

2. If the AT END phrase is specified in the READ statement, any USE procedure associated with the file is not executed. Control is transferred to the imperative statement specified in the AT END phrase. The imperative statement is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement. If control reaches the end of the imperative statement in the AT END phrase, control is transferred to the end of the READ statement.

3. If the AT END phrase is not specified in the READ statement, but an applicable USE procedure is specified either explicitly or implicitly, that procedure is performed and control is transferred to the end of the READ statement.

4. If the AT END phrase is not specified in the READ statement and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit terminates. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the READ statement.

If the at end condition does not arise for the execution of a READ statement, the AT END phrase and its associated imperative statement, if present, are ignored, and the following actions occur in the order shown:

1. If there is a file status data item associated with the file, the appropriate I-O status value is stored into it.

2. If there is an error or exception condition and an applicable USE procedure is specified, either explicitly or implicitly, that procedure is performed and control is transferred to the end of the READ statement.

3. If there is an error or exception condition and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit terminates. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the READ statement.

4. If no error or exception condition exists and a NOT AT END phrase is present, the imperative statement in the phrase is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, control is transferred to the end of the READ statement.

Relative Organization Input-Output

The relative organization input-output statements in the Procedure Division are the CLOSE, DELETE, DELETE FILE, OPEN, READ, REWRITE, START, UNLOCK and WRITE statements.

Function

Relative organization input-output provides the 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 that specifies the logical position of the record in the file.

Organization

Relative file organization is permitted only on mass storage devices (RANDOM, DISK or DISC device in an ASSIGN TO clause).

A relative file consists of records that are identified by relative record numbers. The file may be thought of as comprising a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number, an integer value greater than zero. Records are stored and retrieved based on this number. For example, the 10th record is the one addressed by relative record number 10 and is the 10th record area, whether or not records have been written in the first through the ninth record areas.

Access Modes

In the sequential access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all records that currently exist within the file.

In the random access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing its relative record number in the relative 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 concept used to facilitate specification of the next record to be accessed within a given file during certain sequences of input-output operations. The concept of the file position indicator has no meaning for a file opened in the output or extend mode. The setting of the file position indicator is affected only by the CLOSE, OPEN, READ and START statements.

I-O Status

If the FILE STATUS clause is included in a file control entry, it defines a two-character file status data item for that file. During the execution of each input-output statement that refers to such a file, the runtime system stores a value into the file status data item. Storage of the value is done before the execution of any associated imperative statement and before any applicable USE procedure is executed. The value can be used by the program to determine the status of that input-output operation. The value that is stored into the file status data item is called the I-O status value.

The I-O status value indicates the status of an input-output operation. It also determines whether an applicable USE procedure should be executed: if one of the conditions listed under the heading “Successful Completion” results, an applicable USE procedure is not executed; if any other condition results, such a procedure may be executed depending on rules stated in the section of this chapter that discusses the USE statement (see page 5-13).

Certain classes of I-O status values indicate critical error conditions. They are the ones that begin with the digits 3, 4 and 9. When such conditions arise, certain system-standard error correction procedures may be tried first, depending on the nature of the problem. If they are not successful in clearing the problem, either a user-specified USE procedure is

executed (if one is applicable) and execution of the program continues, or a runtime error message is produced and execution of the run unit terminates.

Upon completion of the input-output operation the I-O status value expresses one of the following conditions:

- Successful Completion. The input-output statement was executed successfully and no exceptional conditions arose. The left character of the I-O status value is 0 for these cases.
- At End. A sequential READ statement was not executed successfully because of an at end condition. The left character of the I-O status value is 1 for these cases.
- Invalid Key. The input-output statement was not executed successfully because of an invalid key condition. The left character of the I-O status value is 2 for these cases.
- Permanent Error. The input-output statement was not executed successfully because of an error that precludes further processing of the file. The problem could be a violation of an external boundary, or a hardware input-output error such as a data check, parity error, transmission error, and so forth. The left character of the I-O status value is 3 for these cases.
- Logic Error. The input-output statement was not executed successfully because an improper sequence of input-output statements was performed on the file, or because of a violation of a user-defined limit. The left character of the I-O status value is 4 for these cases.
- General Error. The input-output statement was not executed successfully because of a condition that is specified by the right character of the I-O status value. The left character of the I-O status value is 9 for these cases.

It should be noted that the I-O status values specified here differ in many respects from the ones defined in earlier versions of RM/COBOL. The new values comply with ANSI COBOL 1985 whereas the old values comply with ANSI COBOL 1974. In the following list, the old values are shown in square brackets following the new values when the two values are not the same. In situations where it is necessary to preserve compatibility with earlier versions of RM/COBOL in this respect, two courses of action are possible: either modify the text of the source program to use the new set of status values, or make use of the 2 Compile Command Option, which causes the compiler to treat the entire program as an ANSI COBOL 1974 program. That option and the language features it controls are discussed in detail in the Chapter 6, *Compiling*, of the *RM/COBOL User's Guide*.

The following list shows the possible I-O status values that can arise as a result of executing an input-output statement that refers to a relative file:

- Successful Completion
  - I-O Status Value=00. The input-output statement is successfully executed and no further information is available concerning the operation.
  - I-O Status Value=04 [97]. A READ statement executed successfully but the length of the record being processed does not conform to the fixed file attributes for the file.
  - I-O Status Value=05. The input-output statement is successfully executed but the file is not present at the time the input-output statement is executed.
    - For a DELETE FILE statement, the referenced file is not available.
    - For an OPEN statement, the referenced optional file is not present. If the open mode is I-O or extend, the file has been created.
- At End Condition with Unsuccessful Completion
  - I-O Status Value=10. A sequential READ statement is attempted and no next (or previous) logical record exists in the file because the end (or beginning) of the file has been reached, or a sequential READ statement is attempted for the first time on an optional input file that is not present.
  - I-O Status Value=14. A sequential READ statement is attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item specified for the file.
- Invalid Key Condition with Unsuccessful Completion
  - I-O Status Value=22. An attempt is made to write a record that would create a duplicate key in a relative file.
  - I-O Status Value=23. Either an attempt is made to randomly access a record that does not exist in the file, or a START or random READ statement is attempted on an optional input file that is not present.
  - I-O Status Value=24. Either an attempt is made to write beyond the externally defined boundaries of a relative file, or a sequential WRITE statement is attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item specified for the file.

- Permanent Error Condition with Unsuccessful Completion
  - I-O Status Value=30. A permanent error exists and no further information is available concerning the input-output operation.
  - I-O Status Value=35 [94]. A permanent error exists because an OPEN statement with the INPUT, I-O, or EXTEND phrase is attempted on a nonoptional file that is not present.
  - I-O Status Value=37 [90, 95]. A permanent error exists because an OPEN statement is attempted on a file that does not support the open mode specified in the OPEN statement, or a DELETE FILE statement refers to a protected file. For OPEN statements, the possible violations are as follows:
    - 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 that are permitted for a relative file when opened in the I-O mode.
    - The INPUT phrase is specified but the file does not support read operations.
  - I-O Status Value=38 [93]. A permanent error exists because an OPEN or DELETE FILE statement is attempted on a file previously closed with lock.
  - I-O Status Value=39[94]. An OPEN or DELETE FILE statement is unsuccessful because of an incompatibility between the fixed file attributes and the attributes specified for the file in the program.
- Logic Error Condition with Unsuccessful Completion
  - I-O Status Value=41 [92]. An OPEN statement is attempted for a file that is already open, or a DELETE FILE statement is attempted for an open file.
  - I-O Status Value=42 [91]. A CLOSE statement is attempted for a file that is not open.
  - I-O Status Value=43 [90]. A DELETE or REWRITE statement in the sequential access mode is attempted for a file, and the last input-output statement executed for the file was not a successfully executed READ statement.
  - I-O Status Value=44 [97]. A boundary violation exists because of an attempt to write or rewrite a record whose length is longer or shorter than the limits established by the RECORD IS VARYING clause.

- I-O Status Value=46 [96]. A sequential READ statement is attempted on a file open in the input or I-O mode and no valid next record has been established for one of the following reasons:
  - The preceding START statement was unsuccessful.
  - The preceding READ statement caused an at end condition.
  - The preceding READ statement was unsuccessful for some other reason.
- I-O Status Value=47 [90, 91]. A READ or START statement is attempted on a file not open in the input or I-O mode.
- I-O Status Value=48 [90, 91]. A WRITE statement is attempted on a file not open in the I-O, output, or extend mode or on a sequential access file open in the I-O mode.
- I-O Status Value=49 [90, 91]. A DELETE or REWRITE statement is attempted on a file not open in the I-O mode.

- General Error
  - I-O Status Value=93. An OPEN statement is attempted on a file that is not available. The availability of a file is determined by several factors, including the lock mode. See the *RM/COBOL User's Guide* for details on the availability of a file.
  - I-O Status Value=94. An OPEN statement is attempted at a time when there is insufficient available memory to provide the required supplementary input-output areas and control structures, or an OPEN statement is attempted for a file that has an attribute that is not supported, or an OPEN statement is attempted for a file that has file attributes that are inconsistent among themselves.
  - I-O Status Value=98. Defective record structure has been found in the file.
  - I-O Status Value=99. A DELETE, READ, or REWRITE statement is attempted that refers to a record locked by another concurrent user. This I-O status value is returned only when the referenced file has an associated file status data item and there is an applicable USE procedure; when this is not the case, the program waits for the record to become available.

Invalid Key Condition

The invalid key condition can occur as a result of the execution of a DELETE, READ, REWRITE, START or WRITE statement. Details regarding the situations that cause an invalid key condition are presented in the sections of Chapter 6, *Procedure Division Statements*, that explain the individual input-output statements. If an invalid key condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected.

If the invalid key condition exists after the execution of the input-output operations called for by the input-output statement, the NOT INVALID KEY phrase, if specified, is ignored and the following actions occur in the order shown:

1. If there is a file status data item associated with the file, the appropriate I-O status value is stored into it.

2. If the INVALID KEY phrase is specified in the input-output statement, any USE procedure associated with the file is not executed. Control is transferred to the imperative statement specified in the INVALID KEY phrase. The imperative statement is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement. If control reaches the end of the imperative statement in the INVALID KEY phrase, control is transferred to the end of the input-output statement.

3. If the INVALID KEY phrase is not specified in the input-output statement, but an applicable USE procedure is specified either explicitly or implicitly, that procedure is performed and control is transferred to the end of the input-output statement.

4. If the INVALID KEY phrase is not specified in the input-output statement and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit terminates. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the input-output statement.

If the invalid key condition does not exist after the execution of the input-output operations called for by an input-output statement, the INVALID KEY phrase, if specified, is ignored and the following actions occur in the order shown:

1. If there is a file status data item associated with the file, the appropriate I-O status value is stored into it.

2. If there is an error or exception condition other than an invalid key condition and an applicable USE procedure is specified, either explicitly or implicitly, that procedure is performed and control is transferred to the end of the input-output statement.

3. If there is an error or exception condition other than an invalid key condition and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit terminates. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the input-output statement.

4. If no error or exception condition exists and a NOT INVALID KEY phrase is present, the imperative statement in the NOT INVALID KEY phrase is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, control is transferred to the end of the input-output statement.

At End Condition

The at end condition can occur as a result of the execution of a Format 1 READ statement. Details regarding the circumstances that cause an at end condition appear in the discussion of the Format 1 READ statement that begins on page 6-146.

When the at end condition arises, execution of the READ statement is unsuccessful and the positioning of the file is not changed. The NOT AT END phrase and its imperative statement, if present, are ignored, and the following actions occur:

1. If there is a file status data item associated with the file, the appropriate I-O status value is stored into it.

2. If the AT END phrase is specified in the READ statement, any USE procedure associated with the file is not executed. Control is transferred to the imperative statement specified in the AT END phrase. The imperative statement is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement. If

control reaches the end of the imperative statement in the AT END phrase, control is transferred to the end of the READ statement.

3. If the AT END phrase is not specified in the READ statement, but an applicable USE procedure is specified either explicitly or implicitly, that procedure is performed and control is transferred to the end of the READ statement.

4. If the AT END phrase is not specified in the READ statement and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit terminates. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the READ statement.

If the at end condition does not arise for the execution of a Format 1 READ statement, the AT END phrase and its associated imperative statement, if present, are ignored, and the following actions occur in the order shown:

1. If there is a file status data item associated with the file, the appropriate I-O status value is stored into it.

2. If there is an error or exception condition and an applicable USE procedure is specified, either explicitly or implicitly, that procedure is performed and control is transferred to the end of the READ statement.

3. If there is an error or exception condition and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit terminates. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the READ statement.

4. If no error or exception condition exists and a NOT AT END phrase is present, the imperative statement in the phrase is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, control is transferred to the end of the READ statement.

Indexed Organization Input-Output

The indexed organization input-output statements in the Procedure Division are the CLOSE, DELETE, DELETE FILE, OPEN, READ, REWRITE, START, UNLOCK and WRITE statements.

Function

Indexed organization input-output provides the capability to access records of a mass storage file in either a random or sequential manner. Each record in an indexed organization file is uniquely identified by the value of one or more keys within that record, except when the DUPLICATES phrase is specified for all the keys associated with the file.

Organization

An indexed organization file 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 recorded 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 should, therefore, be unique and must not be changed when updating the record. The value must be unique unless the DUPLICATES phrase is specified in the RECORD KEY clause. When the DUPLICATES phrase is specified in the RECORD KEY clause, the value of the prime record key is not necessarily a unique identifier for a single record; therefore, in this case, the DELETE and REWRITE statements are disallowed in the random access mode and are sequential operations in the dynamic access mode.

Alternate record keys provide alternate means of retrieval for the records of a file. Such keys are named in the ALTERNATE RECORD KEY clause of the file control entry. The value of a particular alternate record key in each record must be unique unless the DUPLICATES phrase is specified in the ALTERNATE RECORD KEY clause.

Access Modes

For indexed organization, the order of sequential access is ascending based on the value of the current key of reference. If a collating sequence is specified for the file it is used in determining the ascending sequence for keys. Any of the keys defined for the file may be established as the current key of reference during the processing of the file. The order of retrieval from a set of records that have duplicate key of reference values is the original order of arrival of those records into that set. The START statement may be used to establish a starting point within an indexed file for a series of subsequent sequential retrievals.

When an indexed file is accessed in random access mode, input-output statements are used to access the records in a programmer-specified order. The programmer specifies the desired record by placing the value of one of its record keys in a record key or an alternate record key data item.

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 concept used to facilitate specification of the next record to be accessed within a given file during certain sequences of input-output operations. The concept of the file position indicator has no meaning for a file opened in the output or extend mode. The setting of the file position indicator is affected only by the CLOSE, OPEN, READ and START statements.

I-O Status

If the FILE STATUS clause is included in a file control entry, it defines a two-character file status data item for that file. During the execution of each input-output statement that refers to such a file, the runtime system stores a value into the file status data item. Storage of the value is done before the execution of any associated imperative statement and before any applicable USE procedure is executed. The value can be used by the program to determine the status of that input-output operation. The value that is stored into the file status data item is called the I-O status value.

The I-O status value indicates the status of an input-output operation. It also determines whether an applicable USE procedure should be executed: if one of the conditions listed under the heading “Successful Completion” results, an applicable USE procedure is not executed; if any other condition results, such a procedure may be executed depending on the rules for the USE statement that begin on page 5-13.

Certain classes of I-O status values indicate critical error conditions. They are the ones that begin with the digits 3, 4 and 9. When such conditions arise, certain system-standard error correction procedures may be tried first, depending on the nature of the problem. If they are not successful in clearing the problem, either a user-specified USE procedure is executed (if one is applicable) and execution of the program continues, or a runtime error message is produced and execution of the run unit terminates.

Upon completion of the input-output operation the I-O status value expresses one of the following conditions:

- Successful Completion. The input-output statement was executed successfully and no exceptional conditions arose. The left character of the I-O status value is 0 for these cases.

- At End. A sequential READ statement was not executed successfully because of an at end condition. The left character of the I-O status value is 1 for this case.

- Invalid Key. The input-output statement was not executed successfully because of an invalid key condition. The left character of the I-O status value is 2 for these cases.

- Permanent Error. The input-output statement was not executed successfully because of an error that precludes further processing of the file. The problem could be a violation of an external boundary, or a hardware input-output error such as a data check, parity error, transmission error, and so forth. The left character of the I-O status value is 3 for these cases.

- Logic Error. The input-output statement was not executed successfully because an improper sequence of input-output statements was performed on the file, or because of a violation of a user-defined limit. The left character of the I-O status value is 4 for these cases.

- General Error. The input-output statement was not executed successfully because of a condition that is specified by the right character of the I-O status value. The left character of the I-O status value is 9 for these cases.

It should be noted that the I-O status values specified here differ in many respects from the ones defined in earlier versions of RM/COBOL. The new values comply with ANSI COBOL 1985 whereas the old values comply with ANSI COBOL 1974. In the following list, the old values are shown in square brackets following the new values when the two values are not the same. In situations where it is necessary to preserve compatibility with earlier versions of RM/COBOL in this respect, two courses of action are possible: either modify the text of the source program to use the new set of status values, or make use of the 2 Compile Command Option, which causes the compiler to treat the entire program as an ANSI COBOL 1974 program. That option and the language features it controls are discussed in detail in Chapter 6, *Compiling*, of the *RM/COBOL User's Guide*.

The following list shows the possible I-O status values that can arise as a result of executing an input-output statement that refers to an indexed file:

- Successful Completion
  - I-O Status Value=00. The input-output statement is successfully executed and no further information is available concerning the operation.
  - I-O Status Value=02. The input-output statement executed successfully, but a duplicate key is detected. 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 statement, the record just written created a duplicate key value for at least one record key for which duplicates are allowed. For a REWRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.
  - I-O Status Value=04 [97]. A READ statement executed successfully but the length of the record being processed does not conform to the fixed file attributes for the file.
  - I-O Status Value=05. The input-output statement is successfully executed but the file is not present at the time the input-output statement is executed.
    - For a DELETE FILE statement, the referenced file is not available.
    - For an OPEN statement, the referenced optional file is not present. If the open mode is I-O or extend, the file has been created.
- At End Condition with Unsuccessful Completion
  - I-O Status Value=10. A sequential READ statement is attempted and no next (or previous) logical record exists in the file because the end (or beginning) of the file has been reached, or a sequential READ statement is attempted for the first time on an optional input file that is not present.
- Invalid Key Condition with Unsuccessful Completion
  - I-O Status Value=21. A sequence error exists for a sequentially accessed indexed file. Either the prime record key value has been changed by the program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file, or the ascending sequence requirements for successive record key values are violated. A sequentially accessed indexed file includes the execution of a REWRITE statement in the dynamic access mode when the DUPLICATES phrase is specified in the RECORD KEY clause.

- I-O Status Value=22. An attempt is made to write or rewrite a record that would create a duplicate record key value for a record key for which the DUPLICATES phrase is not specified.

- I-O Status Value=23. Either an attempt is made to randomly access a record that does not exist in the file, or a START or random READ statement is attempted on an optional input file that is not present.

- I-O Status Value=24. An attempt is made to write beyond the externally defined boundaries of the file.

- Permanent Error Condition with Unsuccessful Completion

  - I-O Status Value=30. A permanent error exists and no further information is available concerning the input-output operation.

  - I-O Status Value=35 [94]. A permanent error exists because an OPEN statement with the INPUT, I-O or EXTEND phrase is attempted on a nonoptional file that is not present.

  - I-O Status Value=37 [90, 95]. A permanent error exists because an OPEN statement is attempted on a file that does not support the open mode specified in the OPEN statement, or a DELETE FILE statement refers to a protected file. For OPEN statements, the possible violations are as follows:

    - 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 that are permitted for an indexed file when opened in the I-O mode.

    - The INPUT phrase is specified but the file does not support read operations.

  - I-O Status Value=38 [93]. A permanent error exists because an OPEN or DELETE FILE statement is attempted on a file previously closed with lock.

  - I-O Status Value=39[94]. An OPEN or DELETE FILE statement is unsuccessful because of an incompatibility between the fixed file attributes and the attributes specified for the file in the program.

- Logic Error Condition with Unsuccessful Completion
  - I-O Status Value=41 [92]. An OPEN statement is attempted for a file that is already open, or a DELETE FILE statement is executed for an open file.
  - I-O Status Value=42 [91]. A CLOSE statement is attempted for a file that is not open.
  - I-O Status Value=43 [90]. A DELETE or REWRITE statement in the sequential access mode is attempted for a file, and the last input-output statement executed for the file was not a successfully executed READ statement. A DELETE or REWRITE statement in the dynamic access mode is attempted for a file that specifies the DUPLICATES phrase in the RECORD KEY clause and the last input-output statement executed for the file was not a successfully executed READ statement.
  - I-O Status Value=44 [97]. A boundary violation exists because of an attempt to write or rewrite a record whose length is longer or shorter than the limits established by the RECORD IS VARYING clause.
  - I-O Status Value=46 [96]. A sequential READ statement is attempted on a file open in the input or I-O mode and no valid next record has been established for one of the following reasons:
    - The preceding START statement was unsuccessful.
    - The preceding READ statement caused an at end condition.
    - The preceding READ statement was unsuccessful for some other reason.
  - I-O Status Value=47 [90, 91]. A READ or START statement is attempted on a file not open in the input or I-O mode.
  - I-O Status Value=48 [90, 91]. A WRITE statement is attempted on a file not open in the I-O, output, or extend mode or on a sequential access file open in the I-O mode.
  - I-O Status Value=49 [90, 91]. A DELETE or REWRITE statement is attempted on a file not open in the I-O mode.

- General Error

  - I-O Status Value=93. An OPEN statement is attempted on a file that is not available. The availability of a file is determined by several factors, including the lock mode. See the *RM/COBOL User's Guide* for details on the availability of a file.

  - I-O Status Value=94. An OPEN statement is attempted at a time when there is insufficient available memory to provide the required supplementary input-output areas and control structures.

  - I-O Status Value=98. An input-output statement is attempted on a file whose index structure or other critical control characters are defective. Either the file being referred to is not an indexed file at all, or it has been damaged in some way since its last usage or creation. See the discussion of the Indexed File Recovery Utility (recover1), in Appendix G, *Utilities*, of the *RM/COBOL User's Guide*, for assistance in restoring a corrupted indexed file.

  - I-O Status Value=99. A DELETE, READ, or REWRITE statement is attempted that refers to a record locked by another concurrent user. This I-O status value is returned only when the referenced file has an associated file status data item and there is an applicable USE procedure; when this is not the case, the program waits for the record to become available.

Invalid Key Condition

The invalid key condition can occur as a result of the execution of a DELETE, READ, REWRITE, START or WRITE statement. Details regarding the situations that cause an invalid key condition are presented in the sections of Chapter 6, *Procedure Division Statements*, that explain the individual input-output statements. If an invalid key condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected.

If the invalid key condition exists after the execution of the input-output operations called for by the input-output statement, the NOT INVALID KEY phrase, if specified, is ignored and the following actions occur in the order shown:

1. If there is a file status data item associated with the file, the appropriate I-O status value is stored into it.

2. If the INVALID KEY phrase is specified in the input-output statement, any USE procedure associated with the file is not executed. Control is transferred to the imperative statement specified in the INVALID KEY phrase. The imperative statement is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that

causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement. If control reaches the end of the imperative statement in the INVALID KEY phrase, control is transferred to the end of the input-output statement.

3. If the INVALID KEY phrase is not specified in the input-output statement, but an applicable USE procedure is specified, either explicitly or implicitly, that procedure is performed and control is transferred to the end of the input-output statement.

4. If the INVALID KEY phrase is not specified in the input-output statement and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit is terminated. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the input-output statement.

If the invalid key condition does not exist after the execution of the input-output operations called for by an input-output statement, the INVALID KEY phrase, if specified, is ignored and the following actions occur in the order shown:

1. If there is a file status data item associated with the file, the appropriate I-O status value is stored into it.

2. If there is an error or exception condition other than an invalid key condition and an applicable USE procedure is specified, either explicitly or implicitly, that procedure is performed and control is transferred to the end of the input-output statement.

3. If there is an error or exception condition other than an invalid key condition and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit terminates. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the input-output statement.

4. If no error or exception condition exists and a NOT INVALID KEY phrase is present, the imperative statement in the NOT INVALID KEY phrase is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, control is transferred to the end of the input-output statement.

At End Condition

The at end condition can occur as a result of the execution of a Format 1 READ statement. Details regarding the circumstances that cause an at end condition appear in the discussion of the Format 1 READ statement that begins on page 6-146.

When the at end condition arises, execution of the READ statement is unsuccessful and the positioning of the file is not changed. The NOT AT END phrase and its imperative statement, if present, are ignored, and the following actions occur:

1. If there is a file status data item associated with the file, the appropriate I-O status value (10) is stored into it.

2. If the AT END phrase is specified in the READ statement, any USE procedure associated with the file is not executed. Control is transferred to the imperative statement specified in the AT END phrase. The imperative statement is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement. If control reaches the end of the imperative statement in the AT END phrase, control is transferred to the end of the READ statement.

3. If the AT END phrase is not specified in the READ statement, but an applicable USE procedure is specified, either explicitly or implicitly, that procedure is performed and control is transferred to the end of the READ statement.

4. If the AT END phrase is not specified in the READ statement and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit is terminated. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the READ statement.

If the at end condition does not arise for the execution of a Format 1 READ statement, the AT END phrase and its associated imperative statement, if present, are ignored, and the following actions occur in the order shown:

1. If there is a file status data item associated with the file, the appropriate I-O status value is stored into it.

2. If there is an error or exception condition and an applicable USE procedure is specified, either explicitly or implicitly, that procedure is performed and control is transferred to the end of the READ statement.

3. If there is an error or exception condition and no applicable USE procedure is specified, a runtime error message is produced and execution of the run unit terminates. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the READ statement.

4. If no error or exception condition exists and a NOT AT END phrase is present, the imperative statement in the phrase is executed according to the rules for each statement encountered in that imperative statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, control is transferred to the end of the READ statement.

File Locking

In runtime environments in which more than one run unit can be running concurrently, the possibility arises that one run unit must deny concurrent access to a file or set of files by other run units. This is accomplished through file locking. There are several methods provided in RM/COBOL to specify file locking.

The LOCK MODE clause in the file control entry may specify the EXCLUSIVE phrase. The LOCK MODE IS EXCLUSIVE clause causes each OPEN statement to open the file in exclusive mode.

The EXCLUSIVE phrase may be specified in the OPEN statement. This causes the OPEN statement to open each file in exclusive mode.

The LOCK phrase may be specified for an individual file in the OPEN statement. This causes the OPEN statement to open that file in exclusive mode.

When the LOCK MODE clause is not specified for a file and the OPEN statement does not specify the EXCLUSIVE or WITH LOCK phrases, whether the OPEN statement opens the file in exclusive or shared mode depends on the environment supporting the file and a configurable default. (See the topic, "File Sharing," and the FORCE-USER-MODE configuration keyword in the *RM/COBOL User's Guide* for additional information.)

A file may be opened in the input mode as a shared or exclusive file when the same file is open only in the shared or exclusive input mode by other run units. The exclusive input mode prevents other run units only from concurrent updates of the file, not from concurrent reading of the file.

When an attempt is made

- to open a file for which some other run unit has the same file open in exclusive extend, exclusive input-output, or exclusive output mode,
- to open a file in input-output or extend mode for which some other run unit has the same file open in exclusive input mode,
- to open a file in exclusive extend or exclusive input-output mode for which some other run unit has the same file open in any mode, or
- to open a file in exclusive input mode for which some other run unit has the same file open in extend, input-output or output mode

the OPEN statement is unsuccessful. The file status data item, if there is one, is set to a value indicating this condition and any applicable USE procedure for the file is executed.

Regardless of lock mode, a file that is open in any mode by another run unit cannot be opened in the output mode.

In shared file environments opening a file in exclusive mode can improve performance of the other input-output statements executed while the file is open. This is because exclusive mode guarantees that no other run unit will update the file while this run unit has the file open. Therefore, physical records can be locally buffered by the run unit when the file is open in exclusive mode, resulting in lower operating system overhead. For files open in the input-output mode, operating system overhead is further reduced since record locking operations are not necessary.

Record Locking

In runtime environments in which more than one user can be running concurrently, the possibility arises that multiple users may wish to access the same file at the same time. In these circumstances, the following sequence of events may occur: user A reads a record from a file, modifies a field within the record, and then rewrites the record. After user A reads the record but before it is rewritten, user B reads the same record from the same file, modifies it and rewrites it. The final contents of the record depend on the sequence in which these operations occur, and this is not predictable since the two users are not coordinated.

To prevent this sort of destructive interference between concurrent users of shared files, RM/COBOL provides record locking facilities. If the LOCK MODE clause is not specified for a file, the default record locking mode for a file opened in a shared input-output mode is automatic single. (See the topic, “File Sharing,” and the FORCE-USER-MODE configuration keyword in the *RM/COBOL User’s Guide* for

additional information.) If the LOCK MODE clause is omitted, the default record locking mode is as described in the *RM/COBOL User's Guide*. There are four record locking modes: automatic multiple, automatic single, manual multiple, and manual single (for more information, see the section on "Record Locking Modes" on page 5-71). Record locking occurs only when the file is open in the shared input-output mode.

Records need not be locked in order to rewrite or delete them. The runtime system will obtain the lock in those cases where it is not already held by the run unit. The record so locked may contain different data than expected because of the action of other run units sharing the file. In addition, the REWRITE or DELETE statement will be unsuccessful if another run unit has deleted that record or if the record is locked by another run unit and the program executing the DELETE or REWRITE statement is such that it does not wait for the record lock to be released. If the record is successfully locked, the record lock is released upon completion of the REWRITE or DELETE statement. It is the application programmer's responsibility to provide appropriate record locking when necessary for data integrity in a shared file environment by use of the READ statement immediately prior to REWRITE or DELETE statements.

When a run unit attempts to hold multiple record locks, either through one of the multiple record locking modes in one file or single record locking modes in more than one file, it is the application programmer's responsibility to avoid deadlock situations. A deadlock situation occurs when run unit 1 holds a lock on record A and repeatedly attempts to lock record B while run unit 2 holds a lock on record B and repeatedly attempts to lock record A. Each application that shares the same files should lock records in the same order and, upon unsuccessfully locking one record in the series, unlock all currently locked records before attempting to lock the records again.

Programs that use record locking may specify both a file status data item and an applicable USE procedure for each file which is possibly shared by other concurrent run units. When this condition is met, the runtime system invokes the USE procedure with the file status data item set to a value of 99 when a record cannot be locked because it is currently locked by another run unit. This can occur for a DELETE or REWRITE statement if these statements are executed without having previously locked the record to be deleted or replaced. When the 99 status occurs, the program can unlock (by use of the UNLOCK statement) any records already successfully locked and then attempt to obtain the required locks again.

Programs that do not specify both a file status data item and an applicable USE procedure for a shared file will cause the runtime system to wait for a record to be unlocked by another run unit before locking the record for this run unit. Such programs should never attempt to hold multiple record locks, either in one logical file or in two or more logical files, since the program cannot recover from potential deadlock situations with other concurrently executing run units that share those files.

Regardless of whether both a file status data item and applicable USE procedure are defined, if the run unit attempts to lock the same record through two different COBOL file-names that refer to the same physical file, the input-output statement will be unsuccessful with an I-O status value of 99.

Note The maximum number of record locks that may be held simultaneously is a system-dependent parameter. An application should not be designed to hold a large number of simultaneous record locks. Special care should be taken when automatic multiple record locking applies to a file because each READ statement without the NO LOCK phrase will obtain another record lock. When a DELETE, READ, or REWRITE statement is executed that attempts to obtain a record lock that exceeds the maximum number of record locks, the statement will complete unsuccessfully.

Record Locking Modes

There are two ways to obtain record locks, either automatically upon execution of a READ statement or manually upon execution of a READ statement that specifies the LOCK phrase.

There are two ways to release record locks, either implicitly such that only a single record is locked by the run unit for the file or explicitly such that multiple records may be locked by the run unit for the file.

These record locking and unlocking methods may be independently combined to give four distinct record locking modes: automatic single, automatic multiple, manual single and manual multiple.

Automatic Record Locking Modes

When the LOCK MODE IS AUTOMATIC clause is specified in the file control entry or is the default applied to the file, then a record is automatically locked when a READ statement without the NO LOCK phrase is executed successfully in the shared input-output mode. The NO LOCK phrase may be specified in the READ statement to suppress this automatic record locking. When automatic record locking applies, the NO LOCK phrase should be specified in READ statements for which it is known that the accessed record will not be updated by a REWRITE statement or deleted by a DELETE statement.

The automatic record locking modes are automatic single and automatic multiple.

Automatic single record locking applies when the LOCK MODE IS AUTOMATIC clause does not specify the MULTIPLE option in the LOCK ON RECORD phrase or when the LOCK MODE clause is omitted and automatic single record locking is the applicable default for the file. In automatic single record locking mode, at most one

record in the logical file is locked by the run unit at any one time because any input-output statement that refers to the file causes any existing record lock to be released.

Automatic multiple record locking applies when the LOCK MODE IS AUTOMATIC clause specifies the LOCK ON MULTIPLE RECORDS phrase or when the LOCK MODE clause is omitted and automatic multiple record locking is the applicable default for the file. Automatic multiple record locking allows the run unit to hold a number of record locks in one file simultaneously. In automatic multiple record locking mode, existing record locks are not released until a CLOSE or UNLOCK statement that refers to the file-name is executed, except that the successful execution of the DELETE statement causes the record lock to be released for the deleted record.

Manual Record Locking Modes

When the LOCK MODE IS MANUAL clause is specified in the file control entry or is the default applied to the file, then a record is manually locked when a READ statement with the LOCK phrase is executed successfully in the shared input-output mode. A READ statement that does not specify the LOCK phrase does not attempt to lock the record accessed. The LOCK phrase may be omitted in a READ statement for which it is known that the accessed record will not be updated by a REWRITE statement or deleted by a DELETE statement.

The manual record locking modes are manual single and manual multiple.

Manual single record locking applies when the LOCK MODE IS MANUAL clause does not specify the MULTIPLE option in the LOCK ON RECORD phrase or when the LOCK MODE clause is omitted and manual single record locking is the applicable default for the file. In manual single record locking mode, at most one record in the logical file is locked by the run unit at any one time because any input-output statement that refers to the file causes any existing record lock to be released.

Manual multiple record locking applies when the LOCK MODE IS MANUAL clause specifies the LOCK ON MULTIPLE RECORDS phrase or when the LOCK MODE clause is omitted and manual multiple record locking is the applicable default for the file. Manual multiple record locking allows the run unit to hold a number of record locks in one file simultaneously. In manual multiple record locking mode, existing record locks are not released until a CLOSE or UNLOCK statement that refers to the file-name is executed, except that the successful execution of the DELETE statement causes the record lock to be released for the deleted record.

Single Record Locking Modes

Single record locking modes are specified by omission of the MULTIPLE option in the LOCK MODE clause of the file control entry or by configuration of single record locking as the default for files not described with the LOCK MODE clause. In single record locking modes, locked records are implicitly released upon execution of any input-output statement that refers to the file. Thus, at most a single record at a time is locked by the run unit for the file. This single record lock moves from record to record as READ statements that obtain a record lock are executed, or is released if any other input-output statement is executed.

The single record locking modes are automatic single and manual single. Automatic single record locking mode is described in “Automatic Record Locking Modes” on page 5-71. Manual single record locking mode is described in “Manual Record Locking Modes” on page 5-72.

Multiple Record Locking Modes

Multiple record locking modes are specified by explicit inclusion of the WITH LOCK ON MULTIPLE RECORDS phrase in the LOCK MODE clause of the file control entry or by configuration of multiple record locking as the default for files not described with the LOCK MODE clause. In multiple record locking modes, locked records are released only upon execution of a CLOSE or UNLOCK statement that refers to the file-name, except that the successful execution of the DELETE statement causes the record lock to be released for the deleted record.

The multiple record locking modes are automatic multiple and manual multiple. Automatic multiple record locking mode is described in “Automatic Record Locking Modes” on page 5-71. Manual multiple record locking mode is described in “Manual Record Locking Modes” on page 5-72.

Interactive Terminal I-O

RM/COBOL supports three distinct modes of transferring data to and from the terminal in an interactive fashion with the terminal operator:

1. Standard-compliant mode. The Format 1 ACCEPT statement and the Format 1 DISPLAY statement are used to communicate with the terminal in the standard-compliant mode. This mode provides no means of controlling the video and audio features available on many CRT-based terminals but it offers the best chance of complete portability across many different implementations of COBOL. In the standard-compliant mode of terminal communication, the hardware device is driven in a plain line-by-line scrolling fashion, as if it were a typewriter.

2. Field-oriented mode. The Format 3 ACCEPT statement and the Format 2 DISPLAY statement are used to communicate with the terminal in the field-oriented mode. The field-oriented mode supports a wide variety of language features that allow the user to control the majority of the video and audio features available on many CRT-based terminals. It also allows the user to place individual fields anywhere on the screen and to control completely the appearance of the entire screen or any subregion of the screen.

3. Screen-oriented mode. The Format 5 ACCEPT statement and the Format 3 DISPLAY statement are used in conjunction with the Screen Section of the Data Division to communicate with the terminal in the screen-oriented mode. This mode provides much the same control over CRT features as does the field-oriented mode. The primary difference between the two is that in the field-oriented mode the added language elements that control CRT features are in the individual ACCEPT or DISPLAY statements, whereas in the screen-oriented mode they are collected together in the Screen Section of the Data Division.

Both the field-oriented and the screen-oriented modes of terminal control are nonstandard extensions to the COBOL language.

The three modes of communicating with the terminal are not intended to be intermixed within a given run unit. The interaction between the three modes is undefined, and intermixing elements from the three modes leads to results that are unpredictable and probably divergent across various implementations of RM/COBOL. A run unit should be planned with one of the modes in mind, and elements of the other modes should be avoided within that run unit.

Sort-Merge

The sort-merge feature 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, after or both before and after the records are ordered by the SORT or after the records have been combined by the MERGE.

Sort-merge provides the facility for sorting one or more files, or combining two or more files, one or more times within a given execution of a program.

The files listed in the USING and GIVING phrases of the SORT and MERGE statements may be of any organization.

No input-output statement may be executed for the file named in the sort-merge file description.

Communication Facility

The communication facility 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.

Message Control System

The implementation of the communication facility requires that an MCS be present in the operating environment of the object program.

The MCS is the logical interface to the operating system under which the object program operates. The primary functions of the MCS are the following:

- To act as an interface between the object program and the network of communication devices, in much the same manner as an operating system acts as an interface between the object program and such devices as card readers, magnetic tapes, mass storage devices and printers.

- To perform line discipline, including such tasks as dial-up, polling and synchronization.
- To perform device-dependent tasks, such as character translations and insertion of control characters, so that the user can create device-independent programs.

The first function, that of interfacing the object program with the communication devices, is the most obvious to the user. In fact, the user may be unaware that the other two functions exist. Messages from communication devices are placed in input queues by the MCS while awaiting disposition by the object program. Output messages from the object program are placed in output queues by the MCS while awaiting transmission to communication devices. The structures, formats and symbolic names of the queues are defined by the user to the MCS at some time prior to the execution of the object program. Symbolic names for message sources and destinations are also defined at that time. The user must specify, in the program, symbolic names that are known to the MCS.

During the execution of an object program, the MCS performs all necessary actions to update the various queues as required.

Object Program

The object program interfaces with the MCS when it is necessary to send data, receive data, or to interrogate the status of the various queues that are created and maintained by the MCS. In addition, the object program may direct the MCS to establish or break the logical connection between the communication device and a specified portion of the MCS queue structure. The method of handling the physical connection is a function of the MCS.

Relationship of the Object Program to the Message Control System and Communication Devices

The interfaces that exist in a communication environment are established by the use of a CD and associated clauses in the Communication Section of the Data Division. There are two such interfaces:

1. The interface between the object program and the MCS.
2. The interface between the MCS and the communication devices.

The source program uses four statements to control the interface with the MCS:

1. The RECEIVE statement, which causes data in a queue to be passed to the object program
2. The SEND statement, which causes data associated with the object program to be passed to one or more queues
3. The ACCEPT statement with the MESSAGE COUNT phrase, which causes the MCS to indicate to the object program the number of complete messages in the specified queue structure
4. The PURGE statement, which causes the MCS to eliminate a partial message which has been released by one or more SEND statements

The source program uses two statements to control the interface between the MCS and communication devices:

1. The ENABLE statement, which establishes a logical connection between the MCS and one or more communication devices
2. The DISABLE statement, which breaks a logical connection between the MCS and one or more communication devices

Invoking the Object Program

There are two methods of invoking an object program that makes use of the communication facility:

1. Scheduled initiation
2. MCS invocation

The only operating difference between the two methods is that MCS invocation causes the areas referenced by the symbolic queue and subqueue names in the specified CD to be filled.

Scheduled Initiation of the Object Program

An object program using the communication facility may be scheduled for execution through the normal means available in the operating environment of the program, such as job control language. In that case, the program can use three methods to determine what messages, if any, are available in the input queues:

1. ACCEPT statement with the MESSAGE COUNT phrase
2. RECEIVE statement with a NO DATA phrase
3. RECEIVE statement without a NO DATA phrase (in which case a program wait is implied if no data is available)

Invocation of the Object Program by the Message Control System

It is sometimes desirable to schedule an object communication program only when there is work available for it to do. Such scheduling occurs if the MCS determines what object program is required to process the available message and subsequently causes that program to be scheduled for execution. Each object program scheduled by the MCS establishes a run unit. Prior to the execution of the object program, the MCS places the symbolic queue and subqueue names in the associated data items of the communication description entry that specifies the FOR INITIAL INPUT clause, or the MCS places the symbolic terminal name in the associated data item of the communication description entry that specifies the FOR INITIAL I-O clause.

A subsequent RECEIVE statement directed to that CD will result in the available message being passed to the object program.

Determining the Method of Scheduling

A source program can be written so that its object program can operate with either of the two modes of scheduling. The following technique may be used to determine which method was used to load the object program:

- One CD must contain a FOR INITIAL INPUT clause or a FOR INITIAL I-O clause.
- When the program contains a CD with the FOR INITIAL INPUT clause, the Procedure Division may contain statements to test the initial value of the symbolic queue name in that CD. If it is space filled, the object program was activated by the normal runtime invocation process. If it is not space filled, the MCS has invoked the object program and initialized the data item with the symbolic name of the queue containing the messages to be processed.

- When the program contains a CD with the FOR INITIAL I-O clause, the Procedure Division may contain statements to test the initial value of the symbolic terminal name in that CD. If it is space filled, the object program was activated by the normal runtime invocation process. If it is not space filled, the MCS has invoked the object program and initialized the data item with the symbolic name of the communication terminal that is the source of the message to be processed.

Concept of Messages and Message Segments

A message consists of an arbitrary amount of information, usually character data, whose beginning and end are defined or implied. As such, messages comprise the fundamental but not necessarily the most elementary unit of data to be processed in a communication environment.

Messages may be logically subdivided into smaller units of data called message segments which are delimited within a message by means of end of segment indicators (ESI). A message consisting of one or more segments is delimited from the next message by means of an end of message indicator (EMI). In a similar manner, a group of several messages may be logically separated from succeeding messages by means of an end of group indicator (EGI). When a message or message segment is received by the program, a communication description interface area is updated by the MCS to indicate which, if any, delimiter was associated with the text transferred during the execution of that RECEIVE statement. On output the delimiter, if any, to be associated with the text released to the MCS during the execution of a SEND statement is specified or referenced in the SEND statement. Thus, the presence of these logical indicators is recognized and specified both by the MCS and by the object program; however, no indicators are included in the message text processed by programs.

A precedence relationship exists between the indicators EGI, EMI and ESI. EGI is the most inclusive indicator and ESI is the least inclusive indicator. The existence of an indicator associated with message text implies the association of all less inclusive indicators with that text. For example, the existence of the EGI implies the existence of EMI and ESI.

Concept of Queues

The following discussion applies only when the communication environment is established using a CD without the FOR I-O clause.

Queues consist of one or more messages from or to one or more communication devices. They form the data buffers between the object program and the MCS. Input queues are logically separate from output queues.

The MCS logically places in queues or removes from queues only complete messages. Portions of messages are not logically placed in queues until the entire message is available to the MCS. That is, the MCS does not pass a message segment to an object program until all segments of that message are in the input queue, even though the source program uses the SEGMENT phrase of the RECEIVE statement. For output messages, the MCS does not transmit any segment of a message until all of its segments are in the output queue. The number of messages that exist in a given queue reflects only the number of complete messages that exist in the queue.

The process by which messages are placed into a queue is called enqueueing. The process by which messages are removed from a queue is called dequeueing.

Independent Enqueueing and Dequeueing

It is possible that a message may be received by the MCS from a communication device prior to the execution of the object program. In this case, the MCS enqueues the message in the proper input queue until the object program requests dequeueing with the RECEIVE statement. It is also possible that an object program will cause the enqueueing of messages in an output queue which are not transmitted to a communication device until after the object program has terminated. Two common reasons for such occurrences are as follows:

1. The output queue is disabled.

2. The object program creates output messages at a speed faster than the destination can receive them.

Enabling and Disabling Queues

Usually, the MCS enables and disables queues based on circumstances not necessarily related to the program, such as time of day or message activity. However, the program has the ability to enable and disable queues itself by using the ENABLE and DISABLE statements.

A key is required in both statements in order to prevent indiscriminate use of the facility by a user who is not aware of the total network environment, and who may, therefore, disrupt system functions by the untimely issuance of ENABLE and DISABLE statements. However, this action never interrupts a transmission.

Queue Hierarchy

In order to control more explicitly the messages being enqueued and dequeued, it is possible to define in the MCS a hierarchy of input queues, that is, queues comprising queues. Four levels of queues are available. In order of decreasing significance, the queue levels are named queue, sub-queue-1, sub-queue-2 and sub-queue-3.

Chapter 6: Procedure Division Statements

This chapter presents detailed information on the syntax and meaning of each Procedure Division statement. Each Procedure Division statement within a series of statements may be connected to the next by the optional word THEN.

$$\left\{ \textit{imperative-statement-1} \ \text{THEN} \right\} \cdots \ \left\{ \begin{array}{l} \textit{imperative-statement-2} \\ \textit{conditional-statement-1} \end{array} \right\}$$

ACCEPT . . . FROM Statement

The ACCEPT . . . FROM statement causes low volume data to be made available to the specified data item.

Format 1: Accept From System-Name

```
ACCEPT identifier-1 [ FROM { mnemonic-name-3       } ] [ END - ACCEPT ]
                           { low-volume-I-O-name-1 }
```

Data is transferred from the standard input device into the data item referred to by *identifier-1*. The FROM phrase may affect which input device is used. If *mnemonic-name-3* is used in the FROM phrase, it must have been defined in the SPECIAL-NAMES paragraph of the Environment Division with the *low-volume-I-O-name-1* IS *mnemonic-name-3* clause. The associated *low-volume-I-O-name-1* must be CONSOLE or SYSIN.

Note If *identifier-1* is numeric or justified right and the FROM phrase is not specified, the Format 1 ACCEPT statement is treated as if it were a Format 3 ACCEPT statement with the CONVERT phrase. A compiler option suppresses this modification. See the discussion of the Compile Command in Chapter 6, *Compiling*, in the *RM/COBOL-85 User's Guide*, for details.

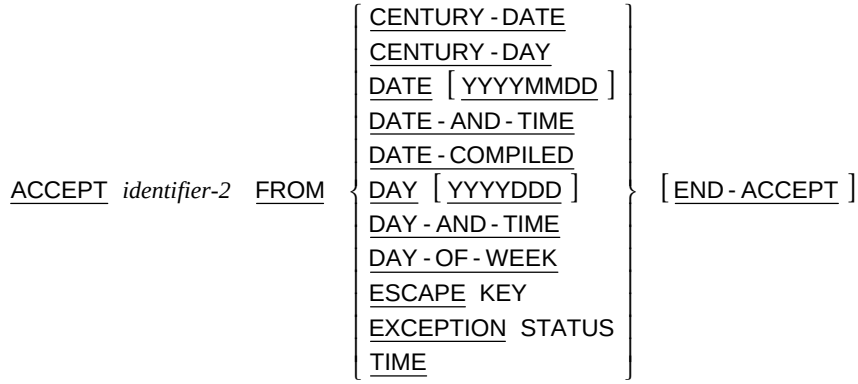
If the size of the receiving data item—or the portion of the receiving data item not yet occupied by transferred data—exceeds the size of the transferred data, the transferred data is stored aligned to the left in the receiving data item (or that portion not yet occupied), and additional data is accepted from the keyboard.

If the size of the transferred data exceeds the size of the receiving data item—or the portion of the receiving data item not yet occupied by transferred data—only the leftmost characters of the transferred data are stored in the receiving data item (or the remaining portion). The remaining characters of the data that do not fit into the receiving data item are discarded.

ACCEPT . . . FROM CONSOLE is treated as if CONSOLE IS CONSOLE was specified in the SPECIAL-NAMES paragraph if CONSOLE has not been otherwise defined.

ACCEPT . . . FROM SYSIN is treated as if SYSIN IS SYSIN was specified in the SPECIAL-NAMES paragraph if SYSIN has not been otherwise defined.

Format 2: Accept From Implicit Definition



For any single Format 2 ACCEPT statement execution, the runtime ensures the consistency of the data returned for cases when the result might be affected by a boundary condition. For example, the runtime guarantees for the DATE-AND-TIME option that the time and date agree when the time is just before or just after midnight. On the other hand, when the DATE and TIME options are used in separate ACCEPT statements near midnight, the program will obtain an inconsistent set of values that is nearly 24 hours off when considered as a pair if midnight occurs between the two ACCEPT statements.

The information requested is transferred according to the rules of the MOVE statement (see page 6-113). CENTURY-DATE, CENTURY-DAY, DATE, DATE-AND-TIME, DATE-COMPILED, DAY, DAY-AND-TIME, DAY-OF-WEEK, ESCAPE KEY, EXCEPTION STATUS, and TIME are implicitly defined data items and, therefore, are not described in the program.

CENTURY-DATE is made up of the data elements year, month, and day. The sequence is YYYYMMDD; thus, a current date of July 1, 2002 would be expressed as 20020701. CENTURY-DATE, when accessed by a program, behaves as if it had been described as an unsigned elementary numeric integer data item eight digits in length.

CENTURY-DAY is made up of the data elements year and day. The sequence is YYYYDDD; thus, a current date of July 1, 2002 would be expressed as 2002182. CENTURY-DAY, when accessed by a program, behaves as if it had been described as an unsigned elementary numeric integer data item seven digits in length.

DATE without the YYYYMMDD phrase is made up of the data elements year, month, and day. The sequence is YYMMDD; July 1, 1988 would be expressed as 880701. DATE, when accessed by a program, behaves as if it had been described as an unsigned elementary numeric integer data item six digits in length.

DATE with the YYYYMMDD phrase is made up of the same data elements and behaves in the same manner as described for CENTURY-DATE.

DATE-AND-TIME is made up of the data elements year, month, day, hours, minutes, seconds, and hundredths of seconds. The sequence is YYYYMMDDHHMMSShh; thus, a current date and time of July 1, 2002 at 2:41 p.m. would be expressed as 2002070114410000. DATE-AND-TIME, when accessed by a program, behaves as if it had been described as an unsigned elementary numeric integer data item sixteen digits in length.

DATE-COMPILED is made up of the data elements year, month, and day for the date the program compilation started (that is, it is a constant for any particular compilation). The sequence is YYYYMMDD; thus, if the program were compiled on July 1, 2002, this would be expressed as 20020701. DATE-COMPILED, when accessed by a program, behaves as if it had been described as an unsigned elementary numeric integer data item eight digits in length.

DAY without the YYYYDDD phrase is made up of the data elements year and day. The sequence is YYDDD; July 1, 1988 would be expressed as 88182. DAY, when accessed by a program, behaves as if it had been described as an unsigned elementary numeric integer data item five digits in length.

DAY with the YYYYDDD phrase is made up of the same data elements and behaves in the same manner as described for CENTURY-DAY.

DAY-AND-TIME is made up of the data elements year, day, hours, minutes, seconds, and hundredths of seconds. The sequence is YYYYDDDHHMMSShh; thus, a current date and time of July 1, 2002 at 2:41 p.m. would be expressed as 200218214410000. DAY-AND-TIME, when accessed by a program, behaves as if it had been described as an unsigned elementary numeric integer data item fifteen digits in length.

DAY-OF-WEEK is composed of a single data element whose contents represent the day of the week. DAY-OF-WEEK behaves as if it had been defined in the Data Division as an unsigned elementary numeric integer data item one digit in length. The value 1 represents Monday, 2 represents Tuesday, . . ., 7 represents Sunday.

ESCAPE KEY provides access to an encoded value that designates the terminator key that terminated the most recent ACCEPT operation. ESCAPE KEY behaves as if defined in the Data Division as an unsigned, two-digit, numeric integer data item. The value of specific keys is determined by the runtime configuration (see the *RM/COBOL User's Guide*.) The default values for these keys are shown in Table 6-3 on page 6-16.

EXCEPTION STATUS provides access to an encoded value that identifies the type of exception condition that occurred during the preceding pop-up window operation or CALL PROGRAM statement execution. EXCEPTION STATUS behaves as if it had

been described as an unsigned elementary numeric integer data item three digits in length. For a pop-up window operation, the possible values and their meanings are described in the "Pop-Up Window Error Codes" table in Chapter 8, *RM/COBOL Features*, of the *RM/COBOL User's Guide*. For a CALL PROGRAM statement execution, the possible values and their meanings are shown here in Table 6-1.

Table 6-1: EXCEPTION STATUS Values

Value	Meaning
000	Called program completed with no exception.
030	Hardware error.
199	CALL PROGRAM failed.
200	Called program exceeds size limitation.
201	Revision incompatibility in called program.
202	Called program is not a legal COBOL program file.
203	Called program not found.
207	Linkage error.
208	Linkage data too big for available memory.

TIME is made up 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. The sequence is HHMMSShh; thus, 2:41 p.m. would be expressed 14410000. TIME, when accessed by a program, behaves as if it had been described as an unsigned elementary numeric integer data item eight digits in length. The minimum value of TIME is 00000000; the maximum value is 23595999.

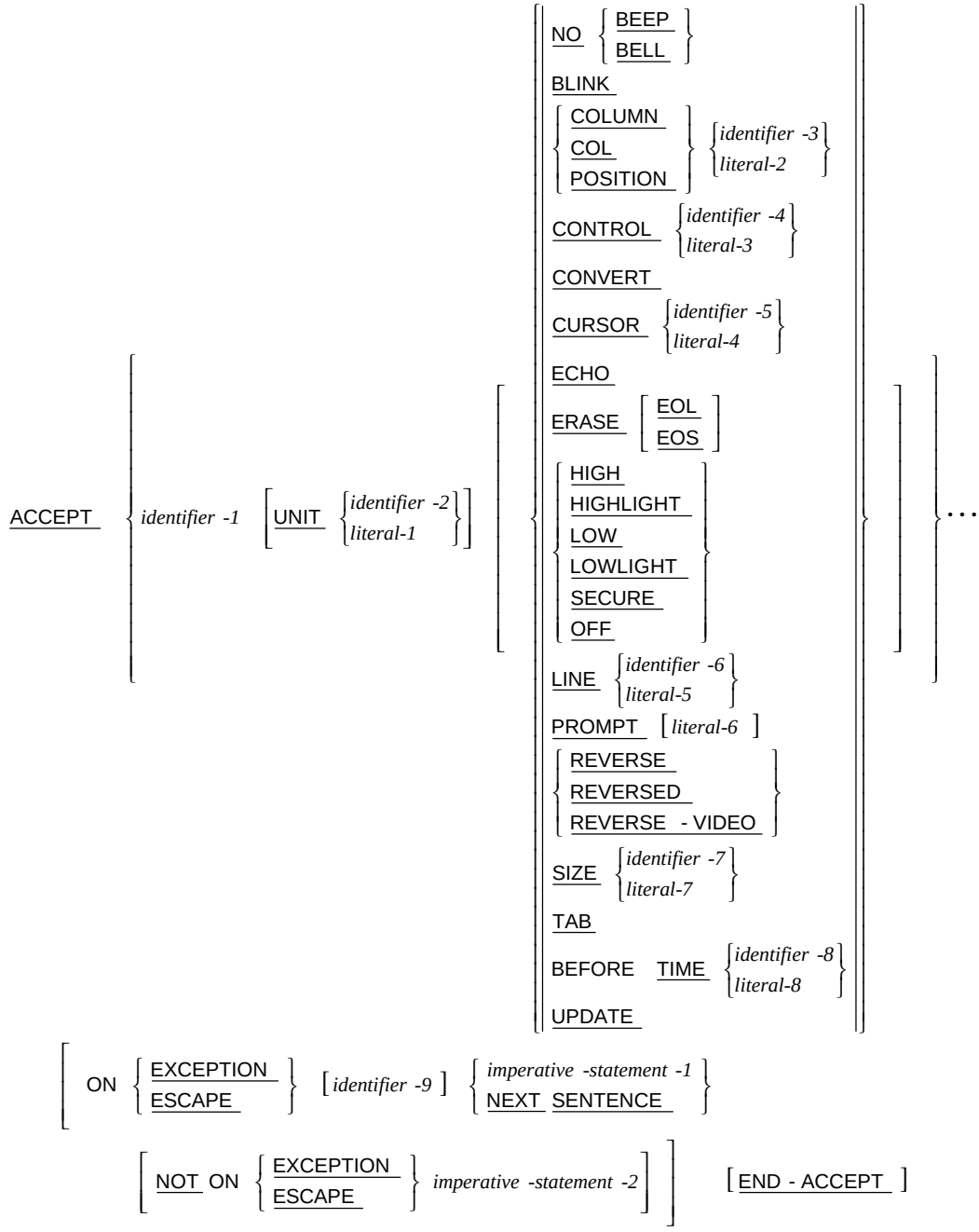
ACCEPT . . . FROM Statement Examples

```
ACCEPT NEXT-ITEM FROM CONSOLE.

ACCEPT continuation-response FROM input-terminal.

ACCEPT YEAR-DAY-VALUE FROM DAY.

ACCEPT TIME-VALUE FROM TIME.

ACCEPT CENTURY-DAY-VALUE FROM CENTURY-DAY.

ACCEPT DATE-AND-TIME-VALUE FROM DATE-AND-TIME.

ACCEPT COMPILATION-DATE FROM DATE-COMPILED.
```

ACCEPT Statement (Terminal I-O)

The terminal I-O ACCEPT statement causes low volume data to be accepted from the terminal keyboard and transferred to the specified data item. ACCEPT statement phrases allow the specification of position, form, and format of the accepted data.

Format 3: Accept Terminal I-O



The Format 3 ACCEPT statement causes the transfer of data from the terminal keyboard. The data replaces the contents of the data item named by *identifier-1*. The receiving data item may have any usage except INDEX.

identifier-2, *identifier-3*, *identifier-5*, *identifier-6*, *identifier-7*, *identifier-8*, and *identifier-9* must be described as integer numeric data items. *literal-1*, *literal-2*, *literal-4*, *literal-5*, *literal-7*, and *literal-8* must be nonnegative integer numeric literals.

identifier-5 must be a nonnumeric data item. *literal-4* must be a nonnumeric literal. *literal-6* must be a nonnumeric literal one character in length.

It is worthwhile to define several terms used to describe the detailed function of each phrase.

- The term "input field" describes a conceptual data item containing the data transmitted from the terminal as displayed on the screen. The size of this data item is determined according to rules outlined in the discussion of the SIZE phrase that begins on page 6-19, and the type of the data item is alphanumeric.
- The term "receiving item" is synonymous with the data item *identifier-1*.
- The term "screen field" applies to the physical field presented on the screen itself.
- The term "field termination" is the means by which the terminal operator indicates the conclusion of data input for an input field; "field termination key" describes a character or character sequence which is interpreted, not as data to be included in the input field, but as field termination. More than one field termination key exists; such keys are differentiated by means of "field termination key codes."

Table 6-2 shows the relationship of the various Format 3 ACCEPT statement phrases to the characteristics of the input field and screen field subject to control by the program.

Note that the CONTROL phrase may be used in many instances to allow dynamic (that is, runtime as opposed to compile time) specification of characteristics.

Features that require support of the host operating system or terminal hardware may not be supported in all circumstances. Unsupported features will compile correctly, but will be ignored at runtime. See the *RM/COBOL User's Guide* for specific details. Also note that some phrases may require that character positions on the screen between fields be reserved for attribute characters (typically, to support the HIGH, LOW, OFF, BLINK, REVERSE, ERASE EOL and ERASE EOS phrases). Take care to allow for attribute characters by not juxtaposing fields that may require them.

Table 6-2: ACCEPT Statement Phrases and Output and Screen Fields

Characteristic	Phrases
Screen field position	LINE, POSITION, ERASE, SIZE, UNIT, CONTROL
Screen field size	*identifier-1*, UPDATE, SIZE, CONTROL
Position within field	CURSOR
Visual attributes	ERASE, HIGH/LOW/OFF, BLINK, REVERSE, CONTROL
Audio attribute	NO BEEP, CONTROL
Default value display	UPDATE, CONTROL
Prompt character fill	PROMPT, CONTROL
Input conversion	UPDATE, CONVERT, CONTROL, *identifier-1*, compiler option
Verification display	ECHO, UPDATE, CONVERT, CONTROL
Field termination	ON EXCEPTION, TAB, CONTROL

When an ACCEPT statement contains more than one receiving operand (*identifier-1*), the values are transferred in the sequence in which the operands are encountered. ACCEPT phrases apply to the previously specified *identifier-1* only. A subsequent *identifier-1* in the same ACCEPT statement is treated as if no previous phrases had been specified (but see the discussion of the POSITION phrase that begins on page 6-17).

An ACCEPT statement may contain no more than one ON EXCEPTION phrase, and if present it must be associated with the last (or only) *identifier-1*.

NO BEEP Phrase

BELL is a synonym for BEEP.

The presence of the NO BEEP phrase in an ACCEPT statement causes suppression of the audio alarm signal.

If the NO BEEP phrase is omitted, an audio alarm signal occurs.

BLINK Phrase

BLINK

The presence of the BLINK phrase causes the PROMPT fill character and any displayed data to be displayed in a blinking mode.

If the BLINK phrase is not specified, the data is displayed in a nonblinking mode.

CONTROL Phrase

CONTROL { *identifier-4* / *literal-3* }

The value of *identifier-4* or *literal-3* in the CONTROL phrase is used to specify a dynamic option list. The value must be a character-string consisting of a series of keywords delimited by commas; some keywords allow assignment of a value by following the keyword with an equal sign and the value. Blanks are ignored in the character-string. Lowercase letters are treated as uppercase letters within keywords. Keywords specified override corresponding static options specified as phrases for the same *identifier-1*. Keywords may be specified in any order. Keywords which specify options that do not apply to the statement are ignored.

The keywords that affect an ACCEPT statement are BEEP, BLINK, CONVERT, ECHO, ERASE, ERASE EOL, ERASE EOS, HIGH, LOW, NO BEEP, NO BLINK, NO CONVERT, NO ECHO, NO ERASE, NO PROMPT, NO REVERSE, NO TAB, NO UNDERLINE, NO UPDATE, OFF, PROMPT, REVERSE, TAB, UNDERLINE, UPDATE and UPPER. The meanings of these keywords when they appear in the value of the CONTROL phrase operand are the same as the corresponding phrases which may be written as static options of the ACCEPT statement, with the addition of the negative forms to allow suppression of statically declared options. The keywords UNDERLINE and UPPER are not available as static options of the ACCEPT statement. When specified, UPPER causes all lowercase alphabetic characters contained in the screen field to be changed to uppercase alphabetic characters before input data conversion and storing in the receiving field. When specified, UNDERLINE causes the field on the screen to be shown in underlined mode, provided the terminal supports that mode. Additional keywords may be supported in environments that have device dependent functions (for example, color control); see the *RM/COBOL User's Guide* for specifics.

The keywords are grouped by function such that only the rightmost appearance in the control value of a keyword from a functional group actually affects the screen field. The functional groupings are as follows:

- Erasure: ERASE, ERASE EOL, ERASE EOS, NO ERASE
- Alarm: BEEP, NO BEEP
- Intensity: HIGH, LOW, OFF
- Blinking: BLINK, NO BLINK
- Video: REVERSE, NO REVERSE
- Termination: TAB, NO TAB
- Prompting: PROMPT, NO PROMPT
- Input data conversion: CONVERT, NO CONVERT
- Output data conversion: UPDATE, NO UPDATE
- Verification: ECHO, NO ECHO
- Input data editing: UPPER
- Underscoring: UNDERLINE, NO UNDERLINE

Note that if the keyword UPDATE is specified, input data conversion is implied; unless *identifier-1* is numeric edited, the keywords CONVERT and NO CONVERT are ignored. In the cases when *identifier-1* is numeric and UPDATE is not specified, NO CONVERT may be used to suppress implicit or explicit input conversion.

CONVERT Phrase

CONVERT

If *identifier-1* is numeric, the CONVERT phrase causes input conversion of the input field to a signed numeric value which is then stored in *identifier-1*. The CONVERT phrase is implied when *identifier-1* is numeric, unless specifically overridden by the NO CONVERT keyword in *identifier-7* or *literal-7* of the CONTROL phrase, or by the use of the compiler option to suppress implied input conversion (see the *RM/COBOL User's Guide* for details).

Numeric input conversion is accomplished by a scan of the input field according to the following rules:

1. Set the sign according to the rightmost sign present in the input data, or positive if no minus sign is present in the input data. The characters CR or DB occurring after all digits in the input field are treated as a minus sign.

2. Set the implied decimal point according to the rightmost period given in the input. If no period is present, the numeric value is an integer. If the DECIMAL-POINT IS COMMA clause was specified in the source program, a comma replaces the period in determining the implied decimal point.

3. Delete all nonnumeric characters from the input field.

See the discussion of the ON EXCEPTION phrase that begins on page 6-14 for more rules which, if violated, cause an error code to be stored in *identifier-8* and an exception condition to exist. A value will be stored in *identifier-1*, however, according to the rules listed above without regard to the presence of an exception condition.

The CONVERT phrase is implied by the UPDATE phrase when *identifier-1* is numeric, but the CONVERT and UPDATE phrases may both be specified without error.

The use of input conversion is strongly recommended for numeric receiving items unless the program needs a different conversion algorithm and performs its own input validation.

If *identifier-1* is numeric and input conversion is not specified (either explicitly or implicitly), *identifier-1* is treated as an elementary alphanumeric data item whose size is equal to the number of data storage positions occupied by *identifier-1*. The data from the unconverted input field is moved to *identifier-1* according to the rules for an alphanumeric move. The use of *identifier-1*, whose value has been set in this manner, in an arithmetic operation will have unpredictable results.

If *identifier-1* is numeric edited and the CONVERT phrase is specified, the input field is converted to a signed numeric value as described above and that value is then stored in *identifier-1* with editing according to the PICTURE character-string for *identifier-1*.

If *identifier-1* is justified right alphanumeric and the CONVERT phrase is specified, the data from the input field is moved to *identifier-1* according to the move rules for a justified right receiving data item.

If *identifier-1* is alphanumeric edited and the CONVERT phrase is specified, those characters in the input field which correspond in position (from the left) to the PICTURE symbols A, X or 9 are moved to their respective positions in *identifier-1*. Spaces will be moved to those positions in *identifier-1* that are represented by the PICTURE symbols A, X or 9 but which have no corresponding positions in the input field. The insertion

characters 0, space and / will be stored in *identifier-1* character positions represented by PICTURE symbols 0, B and /, respectively.

If *identifier-1* is any other type, or if the CONVERT phrase is not specified, the data from the input field is moved to *identifier-1* according to the rules for an alphanumeric move.

CURSOR Phrase

CURSOR { *identifier-5* / *literal-4* }

The value of *identifier-5* or *literal-4* in the CURSOR phrase specifies the initial cursor offset within the screen field from which the data is to be accepted. When *identifier-5* is specified, the cursor offset at field termination is also returned to the program in *identifier-5*.

An offset of 1 represents the leftmost character position of the screen field. A value of zero is treated as 1; a value greater than the size of the screen field is treated as equal to the size of the screen field.

ECHO Phrase

ECHO

The presence of the ECHO phrase in an ACCEPT statement causes the contents of *identifier-1* to be displayed in the screen field following data input. The display of the data is done as if a DISPLAY statement with similar options were executed. Note, however, that the CONVERT phrase in an ACCEPT statement controls only input conversion: output conversion is controlled by the UPDATE phrase.

If *identifier-1* is numeric and input conversion was specified either explicitly or implicitly, the display of the data uses output conversion.

When the ECHO phrase is not specified, the original input data remains in the screen field.

ERASE Phrase

```
ERASE [ EOL ]
      [ EOS ]
```

The presence of the ERASE phrase without either of the reserved words EOL or EOS causes the entire screen of the terminal to be erased. The current line and current position are set to 1.

The presence of the ERASE EOL phrase causes the portion of the line containing the leftmost character of the screen field to be erased from the leftmost character of the screen field to the rightmost character of that line.

The presence of the ERASE EOS phrase causes the portion of the screen to be erased from the leftmost character of the screen field to the rightmost character of the bottom line of the screen.

In all three cases above, erasure occurs before any data is displayed in or accepted from the screen field.

When no ERASE phrase is specified, no erasure occurs before accepting the data.

ON EXCEPTION and NOT ON EXCEPTION Phrases

```
ON { EXCEPTION } [ identifier-9 ] { imperative-statement-1 }
   { ESCAPE    }                  { NEXT SENTENCE         }
      [ NOT ON { EXCEPTION } imperative-statement-2 ]
               { ESCAPE    }
```

ESCAPE is a synonym for EXCEPTION.

The presence of the ON EXCEPTION phrase allows field termination characteristics and conversion errors to be reported.

At field termination, a field termination code is stored in *identifier-9*. This code is normally the code associated with a field termination key. However, if the TAB phrase is not specified for the same *identifier-1*, the field may also be terminated by typing a data character (in other words, not a field termination key) in the rightmost character position of the screen field; this method of field termination results in a value of zero being stored in *identifier-9*. If *identifier-9* is omitted, the value of the termination code may be obtained with a Format 2 ACCEPT statement that specifies the ESCAPE KEY option.

If input data conversion has been specified (see the discussions of the CONVERT and UPDATE phrases that begin on pages 6-11 and 6-22, respectively) and the conversion

process detects a violation of the following rules, the value 98 is stored in *identifier-9*, overriding the field termination key code.

If *identifier-1* is numeric or numeric edited, the following rules are checked:

1. At most one decimal point (period, or comma if DECIMAL-POINT IS COMMA) is allowed.

2. At most one operational sign (+ or – either leading or trailing, or DB or CR as the rightmost nonblank characters of the input field) is allowed.

3. Leading asterisks are allowed, but asterisks may not follow any digits (0 through 9).

4. All characters must be in the set digits (0 through 9), space, period, comma, dollar sign, currency symbol, stroke (/), and plus and minus. In addition, the characters C, R, D, B, and asterisk are allowed as stipulated in rules 2 and 3.

5. The resulting value of the conversion must not cause a size error condition when stored in *identifier-1*.

Note that input data conversion will store a value in *identifier-1* even if one of these rules is violated; see the discussion of the CONVERT phrase that begins on page 6-11 for more details.

If *identifier-1* is alphanumeric edited, an input data conversion error occurs when an input field character in a position corresponding to the PICTURE symbol 0, B, or / (in the PICTURE character-string describing *identifier-1*) exists but is not equal to 0, blank or /, respectively.

When the value of *identifier-9* is nonzero, *imperative-statement-1* may be executed. *imperative-statement-1* will be executed when *identifier-9* has the value 98 (input data conversion rule violation) or the value 99 (input timed out). For other values of *identifier-9*, the execution of *imperative-statement-1* depends on the field termination key; see the *RM/COBOL User's Guide*.

See Table 6-3 for field termination keys and the corresponding field termination codes placed in *identifier-9.*

Table 6-3: Generic Key Names

Generic Key Name [1]	Code	Generic Key Name [1]	Code
Return	13	Function 18	18
Function 1	01	Function 19	19
Function 2	02	Function 20	20
Function 3	03	Command	40
Function 4	04	Attention	41
Function 5	05	Print	49
Function 6	06	Up Arrow	52
Function 7	07	Down Arrow	53
Function 8	08	Home	54
Function 9	09	New Line	55
Function 10	10	Tab Left	56
Function 11	11	Erase Right	57
Function 12	12	Tab Right	58
Function 13	13	Insert Line	59
Function 14	14	Delete Line	61
Function 15	15	Send	64
Function 16	16	Help	83
Function 17	17	Redo	84

[1] *The actual key names for a specific terminal that correspond to the generic key names given above are documented in the* RM/COBOL User's Guide.

HIGH, LOW and OFF Phrases

```
{ HIGH      }
{ HIGHLIGHT }
{ LOW       }
{ LOWLIGHT  }
{ SECURE    }
{ OFF       }
```

HIGHLIGHT is a synonym for HIGH. LOWLIGHT is a synonym for LOW. SECURE is a synonym for OFF.

The presence of the HIGH or LOW phrase causes the PROMPT fill character and the accepted data (if UPDATE, ECHO or both are specified) to be displayed at the specified intensity.

The presence of the OFF phrase causes data to be input from the terminal keyboard but not displayed in the screen field. Space characters are displayed in the screen field in lieu of data characters.

When none of the phrases HIGH, LOW or OFF is specified, the default intensity is HIGH.

LINE and POSITION Phrases

```
LINE { identifier-6 }
     { literal-5    }

{ COLUMN   }   { identifier-3 }
{ COL      } { { literal-2    } }
{ POSITION }   {              }
```

COLUMN and COL are synonyms for POSITION.

The screen field is positioned on the terminal screen by specifying the line and position (that is, the character position within the line) of the leftmost character of the screen field. The top line of the terminal screen is line 1, the line below line 1 is line 2, and so forth. The rightmost character position of a line is immediately followed by the leftmost character position (position 1) of the line below; a screen field may overlap line boundaries on the terminal screen. The leftmost character of the screen field refers to the leftmost character position of that portion of the screen field that is on the topmost line containing a portion of the screen field. Similarly, the rightmost character position of the screen field refers to the rightmost character position of that portion of the screen field that is on the bottommost line containing a portion of the screen field.

The current line and current position prior to the ACCEPT operation for each *identifier-1* may affect the position of the screen field as described in the rules below. At the beginning of a run unit, the current line is the last (bottommost) line and the current position is the leftmost position (position 1) of that line. The current line and current position are changed by each Format 3 ACCEPT and Format 2 DISPLAY operation to be the line and position of the character immediately following the rightmost character of the screen field. If the ERASE phrase (without EOL or EOS) is specified for the same *identifier-1*, the current line and current position are both set to 1.

The value of *identifier-6* or *literal-5* in the LINE phrase specifies the line value for the leftmost character of the screen field. The value of *identifier-3* or *literal-2* in the POSITION phrase specifies the position value for the leftmost character of the screen field.

Determining Line and Position

If the POSITION phrase is omitted, the position value is set to 1 for the first *identifier-1* of a Format 3 ACCEPT statement; this value is also set to 1 if a UNIT phrase is specified for the same *identifier-1*. It is set to zero in all other cases.

If the line value is zero, or if the LINE phrase is omitted, the line value is set according to the following rules:

- If an ERASE phrase (without EOL or EOS) is specified for the same *identifier-1*, the line value is set to 1.
- If the position value is not equal to zero, the line value is set to the current line plus 1.
- If the position value is equal to zero, the line value is set to the current line.

If the position value is greater than the maximum number of characters within a line, the position value is reduced by the maximum number of characters within a line and the line value is incremented by 1. This process is repeated until the position value is not greater than the maximum number of characters within a line.

If the position value is equal to zero, the position value is set to the current position.

If the line value exceeds the number of lines on the screen, the contents of the screen are scrolled up one line and the line value is set to the number of lines on the screen.

If the line of the rightmost character of the screen field exceeds the number of lines on the screen, the contents of the screen are scrolled up the amount of the excess and the line value is reduced by the amount of the excess.

The resulting line value and position value specify the position of the leftmost character of the screen field.

PROMPT Phrase

```
PROMPT [ literal-6 ]
```

The presence of the PROMPT phrase in an ACCEPT statement causes the data to be accepted with prompting. The action of prompting is to display fill characters on the screen in the positions from which data is to be accepted.

The value of *literal-6* specifies the fill character to be used in prompting. *literal-6* must be a single-character, nonnumeric literal or figurative constant. If *literal-6* is omitted in the PROMPT phrase, an underscore is used as the fill character.

When the PROMPT phrase is not specified, the data is accepted without prompting; the original contents of the field on the screen are undisturbed before accepting input unless the UPDATE phrase is specified.

When both the UPDATE and PROMPT phrases are specified, the fill character fills any character positions not occupied by the original value of the receiving operand.

REVERSE Phrase

```
{ REVERSE          }
{ REVERSED         }
{ REVERSE-VIDEO    }
```

REVERSED and REVERSE-VIDEO are synonyms for REVERSE.

The presence of the REVERSE phrase causes any displayed data to be displayed in a reverse video mode.

If the REVERSE phrase is not specified, the data is displayed in normal video mode.

SIZE Phrase

```
SIZE { identifier-7 }
     { literal-7    }
```

The value of *identifier-7* or *literal-7* in the SIZE phrase specifies the size of the screen field and the input field.

If the SIZE phrase is not present, or if a value of zero is specified, the size of the input field and screen field (called the size value) is determined by the characteristics of *identifier-1* and by the presence or absence of input and output conversion (see the

discussions of the CONVERT and UPDATE and phrases that begin on pages 6-11 and 6-22, respectively) as follows:

- If *identifier-1* is numeric and input conversion is specified (either explicitly or implicitly), the size value is set to the number of digits defined in the PICTURE character-string for *identifier-1*, plus 1 if *identifier-1* is signed, plus 1 if *identifier-1* is noninteger.
- If *identifier-1* is numeric, input conversion is not specified, and *identifier-1* is usage DISPLAY, BINARY or equivalent, the size value is set to the number of data storage positions (that is, the number of bytes) occupied by *identifier-1*.
- If *identifier-1* is numeric, input conversion is not specified, the CONVERT phrase is specified, and *identifier-1* is usage PACKED-DECIMAL or equivalent, the size value is set to twice the number of data storage positions occupied by *identifier-1*.
- If *identifier-1* is numeric edited and the UPDATE and CONVERT phrases are specified explicitly, the size value is set as described in rule 1. Note that the number of digits defined in the PICTURE character-string does not include the insertion symbol 0.
- In all other circumstances the size value is set to the number of data storage positions occupied by *identifier-1*.
- If *identifier-1* is numeric and the SIZE phrase is present and the value of its operand is greater than the number of 9's in the PICTURE character-string of *identifier-1*, truncation of the entered value may occur and no conversion error is produced. If the usage of *identifier-1* is COMP-1, binary truncation is done. If the usage of *identifier-1* is COMP-1 and its PICTURE character-string specifies five or more 9's, the entered value is also truncated.

TAB Phrase

TAB

The presence of the TAB phrase in an ACCEPT statement causes a wait for a field termination key; the field termination key will signal field termination.

If the TAB phrase is omitted, field termination occurs when either the end of the input field is encountered or a field termination key is pressed.

TIME Phrase

BEFORE TIME { *identifier-8* | *literal-8* }

The value of *identifier-8* or *literal-8* in the TIME phrase specifies the length of time to wait before automatically terminating when no data is entered during the execution of the ACCEPT statement. The time period is specified in hundredths of seconds, but should be considered only an approximate value because of system variations. For example, a value of 6000 specifies an approximate timeout value of one minute.

A time-out value of 0 indicates that the ACCEPT should terminate immediately if there is no character waiting. A time-out value greater than 4,294,967,295 (a PIC 9(10) data item set to a value 9999999999 is recommended) indicates that the TIME phrase is being overridden and the ACCEPT will behave as if the TIME phrase were not specified.

If the user enters any data during the execution of an ACCEPT statement prior to the completion of the timing interval, the timer is canceled. The user may then take any amount of time to complete the entry of data for the ACCEPT statement as if the TIME phrase had not been specified.

If the timing interval completes without any entry of data by the user, then the ACCEPT statement terminates and returns a value as if the user had typed the Return key, except that an exception condition is raised and the exception status code value 99 is returned.

UNIT Phrase

UNIT { *identifier-2* | *literal-1* }

The UNIT phrase, if specified, must be the first phrase entered. The other phrases may be written in any order.

The value of *identifier-2* or *literal-1* in the UNIT phrase specifies the terminal from which the data is to be accepted. If the UNIT phrase is omitted, the terminal that started the run unit is used.

The UNIT phrase may be ignored by some runtime implementations except as it affects the default value of the POSITION phrase (described previously). This will occur in all systems that do not allow use of terminals other than the one associated with the run unit execution.

UPDATE Phrase

UPDATE

The UPDATE phrase causes the current value of *identifier-1* to be displayed in the screen field with output conversion (internal to human-readable form). The operator may then modify the contents of the screen field before indicating field termination. The data in the input field is then stored in *identifier-1* with input conversion (human-readable to internal form; see the discussion of the CONVERT phrase that begins on page 6-11).

In output conversion, the value of a numeric data item is converted to a standardized format according to the output conversion rules for the DISPLAY statement:

1. A leading, separate minus sign is provided for a negative value.
2. An explicit decimal point is provided for a noninteger value. The representation of this explicit decimal point is a period, except that, if the DECIMAL-POINT IS COMMA clause is specified in the source program, a comma is used instead.
3. Digits are left justified with leading zeroes removed.

With the exception of numeric edited data items, nonnumeric data items are not converted before they are displayed (that is, output conversion for nonnumeric data items is a null operation).

If *identifier-1* is numeric edited and both the CONVERT and UPDATE phrases are specified, a numeric value for *identifier-1* is determined according to the rules for the MOVE statement (MOVE numeric-edited TO numeric). That value is then converted to a standardized form according to the rules listed above. If *identifier-1* is numeric edited, the UPDATE phrase is specified, and the CONVERT phrase is not specified, *identifier-1* is treated as a nonnumeric data item and is not converted before display.

Note that output conversion affects only the appearance of the value in the screen field. The contents of *identifier-1* are not changed by output conversion itself.

Output conversion in an ACCEPT statement is controlled by the UPDATE phrase. The UPDATE phrase also implies input conversion (see the discussion of the CONVERT phrase). Unlike the action of the CONVERT phrase in a DISPLAY statement, the CONVERT phrase in an ACCEPT statement does not control output conversion but instead affects input conversion.

ACCEPT Statement (Terminal I-O) Examples

```
ACCEPT ANSWER-1, ANSWER-2.


ACCEPT START-VALUE LINE 1, POSITION K,
  PROMPT, ECHO, CONVERT.


ACCEPT NEXT-N POSITION 0, PROMPT, ECHO.


ACCEPT YEAR, LINE YR-LN, POSITION YR-POS;
  MONTH, LINE MN-LN, POSITION MN-POS.


ACCEPT PASSWORD-VALUE POSITION 0 OFF.


ACCEPT INVENTORY-COUNT;
ON EXCEPTION FUNCTION-CODE
  PERFORM FUNCTION-KEY-PROCEDURE
END-ACCEPT.


ACCEPT command-string
  LINE command-line
  COLUMN command-column
  CURSOR command-cursor-offset
  CONTROL command-control-string.


ACCEPT FIELD-DATA (INX1) LINE FIELD-LINE (INX1)
  COL FIELD-COLUMN (INX1) CONTROL FIELD-CONTROL (INX1).
```

ACCEPT MESSAGE COUNT Statement

The ACCEPT statement with the MESSAGE COUNT phrase causes the number of complete messages in a queue to be made available.

Format 4: Accept Input CD Message Count

ACCEPT *cd-name-1* MESSAGE COUNT [END-ACCEPT]

cd-name-1 must reference an input CD.

The ACCEPT MESSAGE COUNT statement causes the message count data item specified for *cd-name-1* to be updated to indicate the number of complete messages that exist in the queue structure designated by the contents of the data items specified by *data-name-1* (SYMBOLIC QUEUE) through *data-name-4* (SYMBOLIC SUB-QUEUE-3) of the area referenced by *cd-name-1*.

Upon execution of the ACCEPT MESSAGE COUNT statement, 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-11* (MESSAGE COUNT) of the area associated with the communication description entry to be appropriately updated. (See the discussion of the communication description entry that begins on page 4-64.)

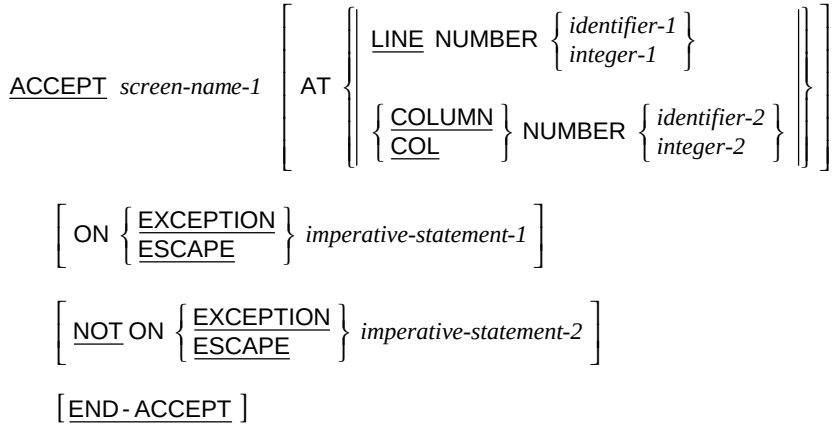
ACCEPT Message Count Statement Example

```
ACCEPT COM-LINE-1 MESSAGE COUNT.
```

ACCEPT Screen-Name Statement

The ACCEPT Screen-Name statement moves data entered by the terminal operator from fields on the terminal screen to data items defined in the Data Division. The organization, placement and visual attributes of the fields on the screen are defined in the Screen Section of the Data Division.

Format 5: Accept Screen-Name



screen-name-1 must be defined as an elementary or group entry in the Screen Section of the Data Division. *identifier-1* and *identifier-2*, when used, must refer to elementary numeric integer data items.

COL is a synonym for COLUMN.

ESCAPE is a synonym for EXCEPTION.

If the LINE phrase is specified, the value of *integer-1* or the current value of the data item referred to by *identifier-1* is used as an increment to each of the explicit or implicit LINE specifications within *screen-name-1*, thus shifting the screen downward the specified number of lines.

A similar rule applies if the COLUMN phrase is specified: the value of *integer-2* or the current value of the data item referred to by *identifier-2* is used as an increment to each of the explicit or implicit COLUMN specifications within *screen-name-1*, thus shifting the screen to the right the specified number of columns.

The following discussion uses the phrase “each input field referred to by *screen-name-1*.” Within the Screen Section, an input field is defined by an elementary entry that contains a

PICTURE clause having the TO or USING option. If *screen-name-1* is an elementary item having a PICTURE clause with a TO or USING option, the phrase "each input field referred to by *screen-name-1*" is a reference to the screen field defined by *screen-name-1*. If *screen-name-1* is a group item, the phrase is a reference to each subordinate elementary input field, taken in the order of their definition.

For each input field referred to by *screen-name-1*, the cursor is positioned at the beginning of the field, the field is filled with the retained value and the operator is given control to enter a new value for that field. If the operator does not wish to change the retained value of the field, the Return key can be used to terminate entry for the field, leaving the value unchanged.

While the operator is entering a value into a field, the local editing keys may be used to revise the value being entered. Until the last input field has been completed, the operator may move the cursor to previously entered input fields to revise their contents. The keys needed to perform these operations are defined in the *RM/COBOL User's Guide*.

The retained value that is shown in the field when the field first becomes active during an ACCEPT operation depends on whether *screen-name-1* has previously been the subject of an ACCEPT or DISPLAY operation within this run unit. If this is the first usage of *screen-name-1* within the run unit, the retained value is ZEROES for numeric items or SPACES for nonnumeric items. If this is not the first usage of *screen-name-1* within the run unit, the retained value is the value left from the last ACCEPT or DISPLAY of *screen-name-1*.

If the current input field is numeric, the operator may enter a leading or trailing sign character (provided the input field allows for a sign) and a decimal point in addition to the numeric digits. The sign and decimal point characters are recognized and removed, using the same de-editing algorithm that is used during a Format 3 ACCEPT of a numeric operand that specifies the CONVERT phrase.

After each input field is completed, the runtime system checks that the characters entered are valid for that particular field. If invalid characters have been entered, an error message is displayed on the bottom line of the screen, and the operator is given control to correct the field. (When the correction operation occurs, existing information on the bottom line, if any, is overlayed and not restored.)

During the course of an ACCEPT operation, the operator may terminate the operation prematurely by use of the Escape key or one of the function keys. If the Escape key is used, the current and all remaining input fields within *screen-name-1* are passed over without changing their retained values. If one of the function keys is used, the current field is completed and becomes the retained value; further input fields, if any, are passed over without changing their retained values. In either case, the escape condition is raised. If neither the Escape key nor one of the function keys is used to signal premature

termination, the ACCEPT operation terminates normally after the last input field has been completed.

The circumstance that caused termination of the ACCEPT operation is recorded by the runtime system, and may be interrogated by executing an ACCEPT . . . FROM ESCAPE KEY statement.

When the ACCEPT operation completes (either normally or prematurely), each retained value corresponding to an input field within *screen-name-1* is moved to its associated item. The move is done according to the standard MOVE rules (which are listed beginning on page 6-113) with the sending item being the retained value as described by the PICTURE clause in the Screen Section entry, and the destination item being the associated item.

If the escape condition is raised during an ACCEPT operation and there is an ON ESCAPE phrase in the ACCEPT statement, control is transferred to *imperative-statement-1* and execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of execution of *imperative-statement-1*, control is transferred to the end of the ACCEPT statement and the NOT ON ESCAPE phrase, if present, is ignored.

If the escape condition is raised during an ACCEPT operation and there is no ON ESCAPE phrase, the escape condition is ignored.

If the escape condition is not raised during an ACCEPT operation and there is a NOT ON ESCAPE phrase, *imperative-statement-1* is ignored, control is transferred to *imperative-statement-2* and execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of execution of *imperative-statement-2*, control is transferred to the end of the ACCEPT statement.

Screen fields within *screen-name-1* that are not input fields have no effect on the operation of the ACCEPT statement.

The BLANK LINE, BLANK REMAINDER and BLANK SCREEN attributes are not active during an ACCEPT operation. The effect of other attributes (AUTO, BELL, BLINK, FULL, REQUIRED, SECURE, and so forth) is as described in Chapter 4, *Data Division*.

The appearance of the screen is undefined and unpredictable when LINE or COLUMN values are specified such that screen fields extend beyond the boundaries of the physical screen, either horizontally or vertically.

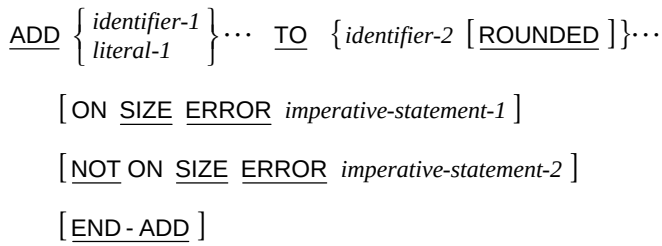
ACCEPT Screen-Name Statement Examples

```
ACCEPT INVOICE-FORM AT LINE 10 COLUMN 5.


ACCEPT EMPLOYEE-RECORD LINE 9
  ON ESCAPE
    DISPLAY ESCAPE-MESSAGE LINE 23
  END-ACCEPT.


ACCEPT EOB-SCREEN AT COL EOB-COL LINE EOB-LINE.
```

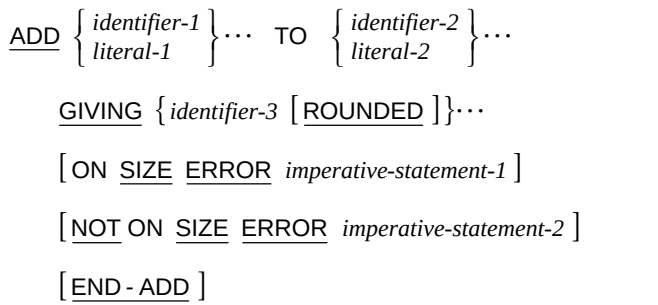
ADD Statement

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

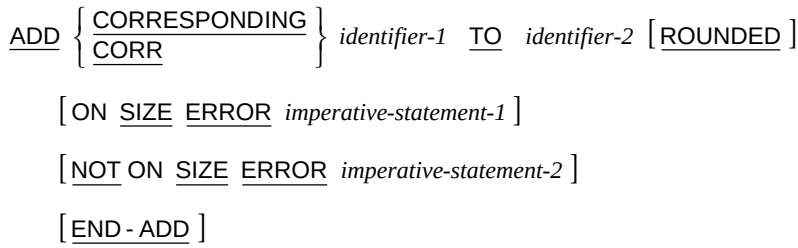
Format 1: Add…To



Format 2: Add…Giving



Format 3: Add Corresponding



In Format 1, the values of the operands preceding the word TO are added together; that sum is then added to the current value of each data item referenced by *identifier-2* storing the result immediately into that data item.

In Format 2, the values of the operands preceding the word GIVING are added together; that sum is then stored as the new value of each data item referenced by *identifier-3*.

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 may refer to either an elementary numeric item or an elementary numeric edited item.

In Format 3, data items in *identifier-1* are added to and stored in the corresponding data items in *identifier-2*.

In Format 3, each identifier must refer to a group item.

Each literal must be a numeric literal.

Additional rules and explanations regarding features of the ADD statement that are common to other arithmetic statements can be found in the discussion of common rules that begins on page 5-16. See in particular the discussions of the ROUNDED phrase, the size error condition, overlapping operands, modes of operation, composite size, and incompatible data.

CORRESPONDING Phrase

```
{ CORRESPONDING } identifier-1  TO  identifier-2 [ ROUNDED ]
{ CORR          }
```

If the CORRESPONDING phrase is used, selected items within *identifier-1* are added to, and the result stored in, the corresponding items in *identifier-2*.

For the ADD statement with the CORRESPONDING phrase:

- The description of *identifier-1* and *identifier-2* must not contain level-number 66, 77, 78, or 88 or the USAGE IS INDEX clause.
- Neither *identifier-1* nor *identifier-2* may be reference modified.
- *identifier-1* or *identifier-2* may be described with the OCCURS or REDEFINES clauses or be subordinate to data items described with the OCCURS or REDEFINES clauses. If *identifier-1* or *identifier-2* is a table element, then the required subscripting must be specified as part of *identifier-1* or *identifier-2*. The specified subscripting will be applied to the selected subordinate corresponding data items, respectively, for *identifier-1* and *identifier-2*.

The rules that govern the selection of eligible subordinate data item pairs are as follows:

1. The data items are not designated by the keyword FILLER and have the same *data-name* and the same qualifiers up to but not including the original group items, *identifier-1* and *identifier-2*.

2. Both of the data items are elementary numeric data items.

3. A data item that is subordinate to *identifier-1* or *identifier-2* 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.

4. The name of each data item that satisfies the above conditions must be unique after application of the implied qualifiers.

If any of the individual operations produces a size error condition, *imperative-statement-1* in the ON SIZE ERROR phrase is not executed until all of the individual additions are completed.

CORR and CORRESPONDING are synonymous.

ADD Statement Examples

```
ADD SALARY TO SALARY.  *>(doubles the value of SALARY)


ADD JOHNS-PAY, PAULS-PAY, ALBERTS-PAY
  GIVING COMPANY-PAY
ON SIZE ERROR
  PERFORM BANKRUPTCY-PROC
END-ADD.


ADD CORRESPONDING
  DAY-TOTALS(DAYX) TO MONTH-TOTALS(MONTHX).


ADD CORR SUB-TOTAL-RECORD TO TOTAL-RECORD ROUNDED
ON SIZE ERROR GO TO ERROR-ROUTINE
NOT ON SIZE ERROR PERFORM AUDIT-ROUTINE
END-ADD.
```

ALTER Statement

The ALTER statement modifies a predetermined sequence of operations.

ALTER { *procedure-name-1* TO [PROCEED TO] *procedure-name-2* }...

procedure-name-1 is the name of a paragraph that contains a single sentence consisting of a GO TO statement without the DEPENDING phrase.

procedure-name-2 is the name of a paragraph or section in the Procedure Division.

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; see the rules for segmentation in Chapter 5, *Procedure Division*.

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.

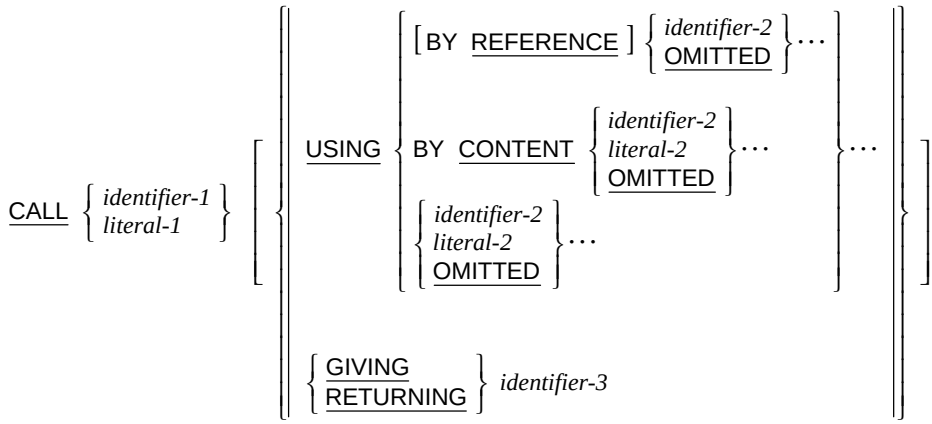
ALTER Statement Examples

```
        ⋮
    PERFORM SET-INITIALIZE-IT.
        ⋮
SWITCH-PARAGRAPH.
    GO TO INITIALIZE-IT.
INITIALIZE-IT.
    INITIALIZE EMPLOYEE-RECORD.
    ALTER SWITCH-PARAGRAPH TO INITIALIZED.
INITIALIZED.
        ⋮
SET-INITIALIZE-IT.
    ALTER SWITCH-PARAGRAPH TO INITIALIZE-IT.
```

CALL Statement

The CALL statement causes control to be transferred from one object program to another, within the run unit.

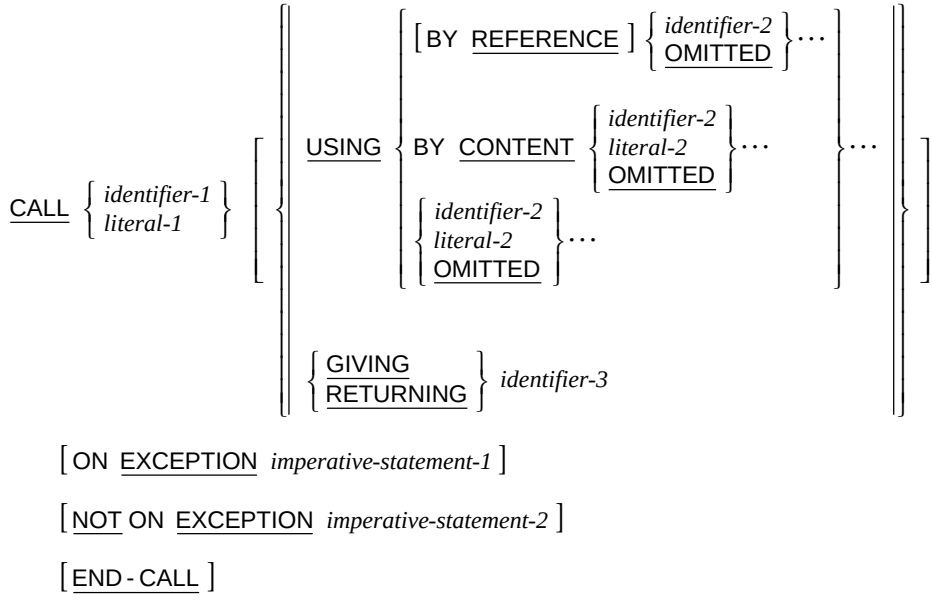
Format 1: Call…On Overflow



[ON OVERFLOW *imperative-statement-1*]

[END-CALL]

Format 2: Call…On Exception



The execution of a CALL statement causes control to pass to the program whose name is specified by the value of *literal-1* or *identifier-1*, termed the "called" program. The program in which the CALL statement appears is the "calling" program.

literal-1 must be a nonnumeric literal.

identifier-1 must be defined as an alphanumeric data item such that its value can be a program-name.

The value of *literal-1* or of the data item referenced by *identifier-1* specifies a program-name that is used to select a program for loading and execution. If the program being called is a COBOL program, the program that is loaded and executed may be selected because the program-name matches the program-name that appears in the PROGRAM-ID paragraph of the called program. For this comparison, any trailing spaces are ignored. Other techniques that are available for selecting the called program are dependent on the runtime operating system, and are, therefore, described in the *RM/COBOL User's Guide*.

Called programs may contain CALL statements. However, a called program must not contain 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 directly or indirectly reference any called program to which control has been transferred and which has not completed execution.

The CALL statement may appear anywhere within a segmented program. When a CALL statement appears in a section with a segment-number greater than or equal to 50, that segment is in its last used state when the EXIT PROGRAM statement returns control to the calling program.

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.

For example, when only two programs in the run unit have the same name as that specified in a CALL statement:

1. One of those two programs must also be contained directly or indirectly either within the separately compiled program which includes that CALL statement or within the separately compiled program which itself directly or indirectly contains the program which includes that CALL statement.
2. The other of those two programs must be a different separately compiled program.

The mechanism used in this example is as follows:

1. If one of the two programs having the same name as that specified in the CALL statement is directly contained within the program which includes that CALL statement, that program is called.
2. If one of the two programs having the same name as that specified in the CALL statement possesses the common attribute and is directly contained within another program which directly or indirectly contains the program which includes the CALL statement, that common program is called unless the calling program is contained within that common program.
3. Otherwise, the separately compiled program is called.

If the called program does not possess the initial attribute, it and each program directly or indirectly contained within it, is in its initial state the first time it is called within a run unit and the first time it is called after a CANCEL to the called program.

On all other entries into the called program, the state of the program and each program directly or indirectly contained within it remains unchanged from its state when last exited.

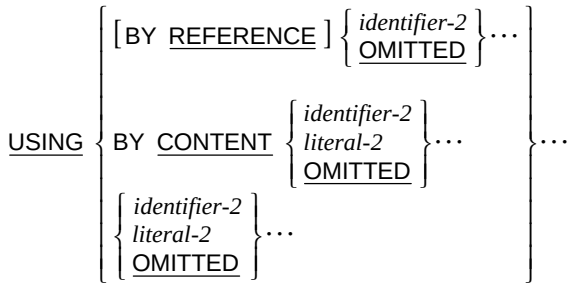
If the called program possesses the initial attribute, it and each program directly or indirectly contained within it, is placed into its initial state every time the called program is called within a run unit.

Files associated with the internal file connectors of a called program are not in the open mode when the program is in an initial state.

On all other entries into the called program, the states and positioning of all such files is the same as when the called program was last exited.

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.

USING Phrase



The operands specified by the USING phrase of the CALL statement indicate those data items available to a calling program that may be referred to in the called program. The order of appearance of the operands in the USING phrase of the CALL statement and the USING phrase in the Procedure Division header is critical. Corresponding operands refer to a single set of data which is available to the called and calling programs. The correspondence is by position, not name. Index-names cannot be made available to the calling program. Index-names in the called and calling programs always refer to separate indexes, except for index-names with the external attribute.

The USING phrase is included in the CALL statement only if there is a USING phrase in the Procedure Division header of the called program. The number of operands in the two USING phrases need not be the same. However, when the two lists have a different number of operands, trailing operands for which there is no corresponding operand in the other list are inaccessible to the called program.

The reserved word OMITTED may be specified to represent an inaccessible operand in the list of operands in the USING phrase of the CALL statement.

In the called program, operands that are inaccessible, either because of omitted trailing operands or use of the word OMITTED in the USING phrase of the CALL statement in the calling program, have a null address. If the called program refers to an inaccessible argument, other than with the ADDRESS special register or in the USING or GIVING phrase of a CALL statement, a data reference error will occur. The called program can check for inaccessible operands by using the ADDRESS special register to test the address of the actual argument for equality to the figurative constant NULL.

Each operand in the USING phrase must have been defined as a data item in the File Section, Working-Storage Section, Communication Section or Linkage Section, and must be a level 01 data item, a level 77 data item, or an elementary data item.

The values of the parameters referenced in the USING phrase of the CALL statement are made available to the called program at the time the CALL statement is executed.

Both the BY CONTENT and BY REFERENCE phrases are transitive across the parameters which follow them until another BY CONTENT or BY REFERENCE phrase is encountered. If neither the BY CONTENT nor the BY REFERENCE phrase is specified prior to the first parameter, the BY REFERENCE phrase is assumed for identifiers and the BY CONTENT phrase is assumed for literals.

Note Prior to v7.5 of RM/COBOL, both identifiers and literals were passed BY REFERENCE when neither the BY CONTENT nor the BY REFERENCE phrase had been explicitly specified. A configuration option has been added to retain the prior behavior; see the SUPPRESS-LITERAL-BY-CONTENT keyword of the COMPILER-OPTIONS configuration record in the *RM/COBOL User's Guide*.

If the BY REFERENCE phrase is either specified or implied for an operand in the USING list, the object program operates as if the associated data item in the called program occupies the same storage area as the corresponding data item in the calling program. The description of the data item in the called program must describe the same number of character positions as described by the corresponding item in the calling program.

If the BY CONTENT phrase is specified or implied for a parameter, the called program cannot change the value of this parameter as referenced in the USING phrase of the CALL statement, though the called program may change the value of the data item referenced by the corresponding data-name in the Procedure Division header of the called program. The data description of each parameter in the BY CONTENT phrase of the CALL statement must be the same (that is, no conversion, extension or truncation) as the data description of the corresponding parameter in the USING phrase of the Procedure Division header.

When the ADDRESS special register is specified in the USING phrase, the result value is always passed by content. The base address of a level 01 or 77 Linkage Section item can only be changed by Formats 5 and 6 of the SET statement. Thus, if the goal is to change the address of a Linkage Section item based on an argument value, the calling program must pass by reference a pointer data item as an argument. The called program may then modify the value of this pointer data item argument. The calling program may then use the pointer data item after the CALL statement in a Format 5 SET statement to set the address of the Linkage Section item.

When an identifier that refers to a level-number 01 or 77 Linkage Section data item that represents a formal argument is specified in the USING phrase of a CALL statement, the data item is resolved according to its description in the calling program. This is an exception to the rule that formal arguments are resolved according to their description in the Linkage Section of the called program. How the data item is resolved mainly affects the length of the data item as seen in the called program. See the "Linkage Section" topic, which begins on page 4-19, for additional information on this special case of resolving Linkage Section record-names.

GIVING Phrase

```
{ GIVING    }
{ RETURNING } identifier-3
```

RETURNING is a synonym for GIVING.

The operand specified in the GIVING phrase of the CALL statement indicates a data item available to a calling program that may be referred to in the called program for purposes of returning a result from the called program. The GIVING argument is functionally the same as any of the USING arguments, except that BY CONTENT and OMITTED are not allowed. The purpose of the GIVING phrase is for source program readability by indicating the return value argument.

The GIVING phrase is included in the CALL statement only if there is a GIVING phrase in the Procedure Division header of the called program. If the GIVING phrase is omitted in the CALL statement when calling a program that includes the GIVING phrase in the Procedure Division header, then the GIVING operand is inaccessible to the called program. If the GIVING phrase is included in a CALL statement where the GIVING phrase is not included in the Procedure Division header of the called program, it has no effect and is ignored. In this case, the operand of the GIVING phrase in the calling program is unchanged after the called program completes.

The ADDRESS special register may not be specified in the GIVING phrase, but an identifier that refers to a pointer data item may be specified.

When an identifier that refers to a level-number 01 or 77 Linkage Section data item that represents a formal argument is specified in the GIVING phrase of a CALL statement, the data item is resolved according to its description in the calling program. This is an exception to the rule that formal arguments are resolved according to their description in the Linkage Section of the called program.

OVERFLOW, EXCEPTION, and NOT EXCEPTION Phrases

ON OVERFLOW *imperative-statement-1*

ON EXCEPTION *imperative-statement-1*

NOT ON EXCEPTION *imperative-statement-2*

If, when a CALL statement is executed, the program specified by the CALL statement can be made available for execution, control is transferred to the called program. After control is returned from the called program, the ON OVERFLOW or ON EXCEPTION phrase, if specified, is ignored and control is transferred to the end of the CALL statement or, if the NOT ON EXCEPTION phrase is specified, to *imperative-statement-2*. In the latter case, execution continues through *imperative-statement-2* according to the rules for each statement specified in that statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the CALL statement.

If, when a CALL statement is executed, it is determined that the program specified by the CALL statement cannot be made available for execution, the NOT ON EXCEPTION phrase, if specified, is ignored and one of the following two actions occurs:

1. If either the ON OVERFLOW or the ON EXCEPTION phrase is specified, control is transferred to *imperative-statement-1*. Execution then continues according to the rules for each statement in *imperative-statement-1*. If a procedure branching or conditional statement that causes explicit transfer of control is encountered, control is transferred in accordance with the rules for that statement, otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the CALL statement.

2. If neither the ON OVERFLOW nor the ON EXCEPTION phrase is specified, a runtime error message is produced and execution of the run unit is terminated.

Reasons for not making a called program available for execution include the following:

- It cannot be found using the search patterns, which are described in the *RM/COBOL User's Guide*.
- Its format is not one of the legal formats for an RM/COBOL called program.
- There is insufficient available memory to load it.

CALL Statement Examples

```
0010.
    IF CHOICE-1 = "01"  MOVE "APP01" TO SUBPRG1
    ELSE IF CHOICE-1 = "02" MOVE "APP02" TO SUBPRG1
    ELSE PERFORM 0020-RETRY-CHOICE GO TO 0010
    END-IF END-IF.
       ⋮
    CALL SUBPRG1.  *>Call "APP01" or "APP02" per choice.


    CALL "REORDER" USING TABLE1 GIVING TABLE1-TOTAL.


 RETRY-1.
    CALL SUBNAME OF SUBTABLE (IX) GIVING STATUS-1
      USING FUNCTION-TYPE, ITEM-1, ITEM-2,
    ON EXCEPTION PERFORM CANCEL-PARAGRAPH GO TO RETRY-1
    NOT ON EXCEPTION SET SUB-LOADED (IX) TO TRUE
    END-CALL.


    CALL "C$SCRD" USING
      SCREEN-BUFFER, OMITTED, SCREEN-LINE, SCREEN-COLUMN.
```

CALL PROGRAM Statement

The CALL PROGRAM statement transfers control from the current program to another program, with implicit termination of the current program and no expectation of return.

```
CALL PROGRAM { identifier-1 } [ USING { identifier-2 } ... ]
             { literal-1    }         { literal-2    }
                                      { OMITTED      }

[ ON EXCEPTION imperative-statement-1 ]

[ END-CALL ]
```

literal-1 must be a nonnumeric literal.

identifier-1 must be defined as an alphanumeric data item such that its value can be a program-name.

If the program specified by *literal-1* or by the current value of the data item identified by *identifier-1* can be found and loaded, the USING operands, if any, are copied to a save area in memory, the current run unit is canceled, and control is transferred to the specified program, passing the saved USING operands as parameters.

Cancellation of the run unit in which the CALL PROGRAM statement is executed includes closing any files that are in an open mode and the release of all external objects.

The specified program is entered as the main program of a completely new run unit in the same way as a program started from the command line, except that the argument list to this program is not restricted in the same way; see the *RM/COBOL User's Guide* for an explanation of the restrictions on the argument list for a main program started from the command line. The main program of a run unit started with the CALL PROGRAM statement may receive all the arguments passed by that CALL PROGRAM statement.

The specified program is not under the control of a calling program. There is no provision for return of control from the specified program to the program in which the CALL PROGRAM statement is executed. If the specified program executes an EXIT PROGRAM statement, execution of the program continues with the next executable statement.

If the program referred to by *literal-1* or by the current value of the data item identified by *identifier-1* cannot be found or loaded, the exception condition is raised and control remains in the current program.

If the exception condition is raised and there is an ON EXCEPTION phrase, control is transferred to *imperative-statement-1* and execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of execution of *imperative-statement-1*, control is transferred to the end of the CALL PROGRAM statement.

The reason for the exception condition can be determined by executing an ACCEPT . . . FROM EXCEPTION STATUS statement.

If the exception condition is raised and there is no ON EXCEPTION phrase, the exception condition is ignored.

Selection of the program to be activated by a CALL PROGRAM statement is done using the same rules as are used for that purpose by the CALL statement, described on page 6-33.

The USING phrase is subject to the same conditions and has the same purpose and effect as described previously for the CALL statement.

CALL PROGRAM Statement Examples

```
CALL PROGRAM "MENU2" USING COMMON-DATA
ON EXCEPTION
  DISPLAY "Chain to MENU2 failed."
  STOP RUN
END-CALL.

CALL PROGRAM CHAIN-NAME USING ARGUMENT-AREA
ON EXCEPTION
   ACCEPT EX-STATUS FROM EXCEPTION STATUS
   PERFORM 0030-CHAIN-ERROR-STATUS
   STOP RUN
END-CALL.
```

CANCEL Statement

The CANCEL statement ensures that the next time the referenced programs are called they will be in their initial state. For a separately compiled program, the CANCEL statement releases the memory areas occupied by the referenced programs.

CANCEL { *identifier-1* / *literal-1* } ...

literal-1 must be a nonnumeric literal.

identifier-1 must be defined as an alphanumeric data item such that its value can be a program-name.

Subsequent to the execution of a CANCEL statement, the programs it refers to cease to have a logical relationship to the run unit in which the CANCEL statement appears. A subsequently executed CALL statement naming such a program results in that program being initiated in its initial state. The memory areas associated with the named programs are released so as to be made available for disposition by the runtime system.

A program named in a CANCEL statement in another program must be callable by that other program.

When an explicit or implicit CANCEL statement is executed, all programs contained within the program referenced by the CANCEL statement are also canceled. The result is the same as if a valid CANCEL statement were executed for each contained program in the reverse order in which the programs appear in the separately compiled program.

A program named in the CANCEL statement must not refer to any program that has been called and has not yet executed an EXIT PROGRAM statement.

A logical relationship to a canceled subprogram is established only by execution of a subsequent CALL statement. A called program is canceled either by being referred to as the operand of a CANCEL statement, by the termination of the run unit of which the program is a member or by execution of an EXIT PROGRAM statement in a called program that possesses the initial attribute (see the discussion of the PROGRAM-ID paragraph that begins on page 2-2).

No action is taken when a CANCEL statement is executed naming a program that has not been called in this run unit or has been called and is at present canceled. Control passes to the next statement.

The contents of data items in external data records described by a program are not changed when that program is canceled.

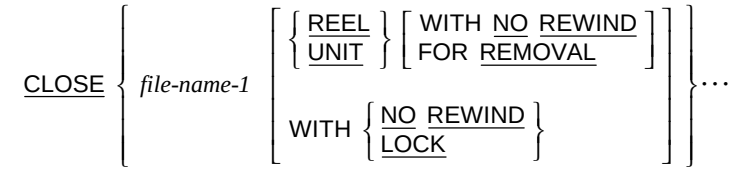
During execution of an explicit or implicit CANCEL statement, an implicit CLOSE statement without any optional phrases is executed for each file in an open mode that is associated with an internal file connector in the program named in the explicit CANCEL statement or implied in the implicit CANCEL statement. Any USE procedures associated with any of these files are not executed.

CANCEL Statement Examples

```
CANCEL "SUB01", "SUB02".


CANCEL SUBPROGRAM-NAME-HOLDER.


CANCEL-PARAGRAPH.
     SET SUB-UNLOADED TO FALSE.
     PERFORM VARYING IX FROM 1 BY 1 UNTIL IX > 4
       IF SUB-LOADED OF SUBTABLE (IX)
         CANCEL SUBNAME OF SUBTABLE (IX)
         SET SUB-LOADED OF SUBTABLE (IX) TO FALSE
         SET SUB-UNLOADED TO TRUE
       END-IF
     END-PERFORM.
     IF NOT SUB-UNLOADED
       DISPLAY "Insufficient memory."
       STOP RUN
     END-IF.
```

CLOSE Statement

The CLOSE statement terminates the processing of reels or units, and files with optional rewind, lock, or both, or removal where applicable.



The files referenced in the CLOSE statement need not all have the same organization or access.

The NO REWIND, REEL, and UNIT phrases may only be specified for files that are sequential organization.

The function of a CLOSE statement (with no options) is to cause the runtime system to close the file. For files opened for OUTPUT, the runtime system also writes an EOF as it closes the file.

If a STOP RUN statement is executed prior to closing the file, the runtime system closes the file. Such a close is equivalent to the execution of a CLOSE statement except that any associated USE procedure is not executed.

A CLOSE statement may be executed only for a file in an open mode.

Once a CLOSE statement without the REEL or UNIT phrase has been executed for a file, no other statement (except the SORT or MERGE statement with the USING or GIVING phrase) can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

The execution of a CLOSE statement causes the value of the file status data item, if any, associated with *file-name-1* to be updated.

REEL and UNIT Phrases

```
{ REEL }
{ UNIT }
```

The REEL and UNIT phrases are synonymous in this context. The CLOSE REEL and CLOSE UNIT statements cause processing to be discontinued on the current volume and to be continued on the next volume of a multivolume series. CLOSE REEL and CLOSE UNIT on a single-volume file are ignored.

The REEL and UNIT phrases may only be specified for sequential organization files.

The action of the phrase for multivolume disk files and tape files depends on the open mode:

- For files which are open OUTPUT, the current volume is closed. The next WRITE statement will cause the record to be written to the next volume in the series. If no next volume is described or available to the series, an error occurs.
- For files which are open INPUT or I-O, the current volume is closed. The next READ statement will obtain the first record from the next volume in the series. If no next volume exists for the file, the next READ statement causes an at end condition.

NO REWIND Phrase

```
WITH NO REWIND
```

The NO REWIND phrase may be used to write and read multiple files on a tape with a single file-name. The phrase suppresses the automatic rewinding of a tape volume when a CLOSE statement, without the NO REWIND phrase, is executed.

The NO REWIND phrase may only be specified for sequential organization files.

Following a CLOSE *file-name-1* WITH NO REWIND, an OPEN *file-name-1* WITH NO REWIND may be used to write or read the next file on the tape. For input, the file must be closed without rewinding after reading all the records in the file; otherwise, the open without rewinding will fail since the tape is not positioned at the beginning of a file.

The NO REWIND phrase is ignored for files that are not on tape or directed to a printer.

Specifying both the UNIT or REEL phrase and the NO REWIND phrase for a single file-name within a CLOSE statement is an allowed syntactical form, but in such a case the NO REWIND phrase has no meaning and is ignored at execution time.

REMOVAL Phrase

FOR REMOVAL

The REMOVAL phrase may be used so that the operating system is notified that the reel or unit is logically removed from this run unit. However, the reel or unit may be accessed again, in its proper order of reels and units within the file, if a CLOSE statement without the REEL or UNIT phrase is subsequently executed for this file followed by the execution of an OPEN statement for the file.

The REMOVAL phrase may only be specified for sequential organization files.

The NO REWIND and REMOVAL phrases have no effect at object time if they do not apply to the storage medium on which the file resides.

LOCK Phrase

WITH LOCK

The function of the CLOSE WITH LOCK statement is to perform the CLOSE function and set a flag to prevent the file from being opened again during execution of this program. In some runtime environments, the CLOSE WITH LOCK statement frees system resources that would otherwise not be freed until the run unit terminates.

The execution of a CLOSE statement always releases any file lock or record locks held by the run unit for *file-name-1*. The LOCK phrase of the CLOSE statement is unrelated to file locking and record locking.

CLOSE Statement Examples

```
CLOSE TRANSACTION-FILE.

CLOSE LOG-FILE WITH LOCK, PRINT-FILE.

CLOSE INPUT-FILE REEL FOR REMOVAL.

CLOSE TAPE-FILE-1 WITH NO REWIND.

CLOSE DATA-BASE WITH LOCK.
   ⋮
OPEN I-O DATA-BASE.
IF DB-STATUS = "38"
  DISPLAY "Data-base file closed with lock."
  STOP RUN
END-IF.
```

COMPUTE Statement

The COMPUTE statement calculates the value of an arithmetic expression and assigns the value to one or more data items.

```
COMPUTE { identifier-1 [ ROUNDED ] }... = arithmetic-expression-1

    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ]

    [ END-COMPUTE ]
```

identifier-1 must refer to either an elementary numeric item or an elementary numeric edited item.

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.

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite of operands, receiving data items, or both, imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY and DIVIDE.

Additional rules and explanations regarding features of the COMPUTE statement that are common to other arithmetic statements can be found in the discussion of common rules that begins on page 5-16. See in particular the discussions of the ROUNDED phrase, the size error condition, overlapping operands, modes of operation, composite size, and incompatible data.

COMPUTE Statement Examples

```
COMPUTE SALARY ROUNDED = WAGES * REGULAR-HOURS
  + WAGES * OVERTIME-HOURS * 1.5.


COMPUTE SECONDS = (((HRS * 60) + MIN) * 60) + SEC
ON SIZE ERROR
  DISPLAY "Time value out of range."
  STOP RUN
END-COMPUTE.


COMPUTE AVERAGE = TOTAL-1 / COUNT-1
ON SIZE ERROR MOVE 0 TO AVERAGE END-COMPUTE.


COMPUTE INTEREST-PER-PERIOD ROUNDED =
  INTEREST-APR / 1200.
COMPUTE PAYMENT-RND ROUNDED PAYMENT-TRUNC =
  (INITIAL-PRINCIPAL * INTEREST-PER-PERIOD) /
  (1 - (1 + INTEREST-PER-PERIOD) **
  (- NUMBER-OF-PERIODS)).
```

CONTINUE Statement

The CONTINUE statement has no effect on the execution of the program.

CONTINUE

The CONTINUE statement may be used anywhere a conditional statement or an imperative statement may be used.

The CONTINUE statement is most useful within a conditional phrase of another statement when no action is desired when the condition occurs.

CONTINUE Statement Examples

```
CONTINUE.


IF NORMAL-RESULT = "Y"
  CONTINUE
ELSE
  PERFORM EXCEPTION-CASE-ANALYSIS
END-IF.


ACCEPT PART-DESCRIPTION UPDATE ERASE EOL
  ON EXCEPTION EXCP-CODE CONTINUE END-ACCEPT.
```

DELETE Statement (Relative and Indexed I-O)

The DELETE statement logically removes a record from a mass storage file.

```
DELETE  file-name-1  RECORD

    [ INVALID  KEY  imperative-statement-1 ]

    [ NOT  INVALID  KEY  imperative-statement-2 ]

    [ END - DELETE ]
```

After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

The execution of a DELETE statement does not affect the contents of the record area associated with *file-name-1* or the contents of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with *file-name-1*.

The file referenced by *file-name-1* must be open in the I-O mode at the time of execution of this statement.

For a file in sequential access mode, the last input-output statement executed for *file-name-1* prior to the execution of the DELETE statement must have been a successfully executed READ statement. The runtime system logically removes from the file the record that was accessed by that READ statement.

For a file in random or dynamic access mode, except for an indexed file described with the DUPLICATES phrase in the RECORD KEY clause, the runtime system logically removes from the file the record identified by the contents of the key data item associated with *file-name-1*. If the file does not contain the record specified by the key, the invalid key condition exists. For a relative file, the key data item is the relative key data item specified in the RELATIVE KEY phrase of the ACCESS MODE clause of the file control entry for *file-name-1*. For an indexed file, the key data item is the prime key data item specified in the RECORD KEY clause of the file control entry for *file-name-1*.

For an indexed file described with the DUPLICATES phrase in the RECORD KEY clause, the DELETE statement in the dynamic access mode is executed as if the file were in the sequential access mode and the DELETE statement in the random access mode is not allowed.

The execution of the DELETE statement causes the value of the specified file status data item, if any, associated with *file-name-1* to be updated.

The file position indicator is not affected by the execution of a DELETE statement.

The INVALID KEY phrase and the NOT INVALID KEY phrase must not be specified for a DELETE statement that references a file which is in sequential access mode.

The INVALID KEY phrase must be specified for a DELETE statement that references a file which is not in sequential access mode and for which an applicable USE procedure is not specified.

See the section on relative organization input-output (starting on page 5-50) or the section on indexed organization input-output (starting on page 5-59) for additional information on the invalid key condition and the use of the INVALID KEY and NOT INVALID KEY phrases.

The record to be deleted by the execution of the DELETE statement must not be locked by another run unit. For a shared input-output file, the run unit executing the DELETE statement should obtain a record lock by preceding the DELETE statement with a READ statement that locks the record to be deleted. If the run unit does not already hold a lock on the record to be deleted, the runtime system will attempt to obtain the lock. If the lock cannot be obtained because another run unit holds a lock on the record, subsequent action of the program is as described for the READ statement when attempting to lock a record already locked by another run unit. If the lock cannot be obtained because this run unit holds a lock on the record through another COBOL file-name, the DELETE statement is unsuccessful. See “File Locking” on page 5-68 and “Record Locking” on page 5-69 for additional information on coordinating file updates in a shared file environment.

After successful execution of the DELETE statement, any record lock held by the run unit on the deleted record is released regardless of the record locking mode applicable to *file-name-1*.

In single record locking modes when a different record than the one being deleted is locked, that record lock is released upon execution of the DELETE statement.

In multiple record locking modes any record locks held by the run unit for *file-name-1* are not released upon execution of the DELETE statement, except for the record lock on the deleted record.

DELETE Statement Examples

```
DELETE INVENTORY-FILE RECORD; INVALID KEY
  PERFORM BAD-KEY-PROCEDURE END-DELETE.


DELETE STATUS-FILE RECORD.


MOVE DB-DELETE-KEY TO DB-KEY.
DELETE DATA-BASE RECORD
INVALID KEY PERFORM DB-INVALID-KEY-HANDLER
NOT INVALID KEY PERFORM DB-SUCCESS-HANDLER
END-DELETE.
```

DELETE FILE Statement

The DELETE FILE statement causes the removal of the referenced files from the runtime file structure.

DELETE FILE { *file-name-2* }... [END-DELETE]

Each file referred to by *file-name-1* is deleted from the runtime file structure provided the following conditions are all true:

- The file is not in the open mode.
- The file was not previously closed with lock during this execution of the program.
- The file exists.
- The runtime file system supports file deletion.
- The file is not protected from deletion by a mechanism of the runtime file system.
- The fixed file attributes specified for the file match the actual fixed file attributes of the existing file.

For each file referred to by *file-name-1*, the value of its file status data item, if any, is updated.

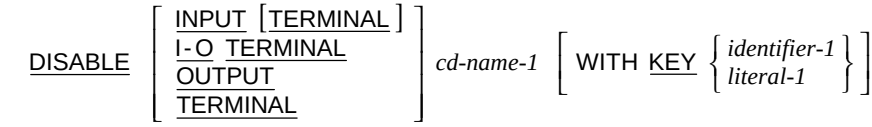
When a DELETE FILE statement references a file that does not exist, the statement executes successfully. Otherwise, a failure of deletion causes the execution of any applicable USE procedure.

DELETE FILE Statement Examples

```
DELETE FILE TEMP-FILE-1 TEMP-FILE-2.

DELETE FILE OLD-TRANSACTION-FILE END-DELETE.
```

DISABLE Statement

The DISABLE statement notifies the Message Control System (MCS) to inhibit data transfer between specified output queues and destinations for output, between specified sources and input queues for input or between the program and one specified source or destination for input-output.



cd-name-1 is defined below for each phrase.

identifier-1 must refer to a data item of category alphanumeric.

literal-1 must be a nonnumeric literal.

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. No action is taken when a DISABLE statement is executed which specifies a source or destination that is already disconnected, except that the value in the status key data item indicates this condition. The logical path for the transfer of data between the object programs and the MCS is not affected by the DISABLE statement.

The MCS ensures that the execution of a DISABLE statement causes the logical disconnection at the earliest time the source or destination is inactive. The execution of the DISABLE statement never causes the remaining portion of the message to be terminated during transmission to or from a terminal.

A DISABLE statement that lacks an INPUT, OUTPUT, I-O or TERMINAL keyword is treated according to the format of the description of the cd-name:

- A DISABLE statement that refers to an INPUT cd-name and does not specify the INPUT keyword is treated as if the INPUT clause without the keyword TERMINAL were specified.
- A DISABLE statement that refers to an OUTPUT cd-name and does not specify the OUTPUT keyword is treated as if the OUTPUT clause were specified.
- A DISABLE statement that refers to an I-O cd-name and does not specify the I-O keyword is treated as if the I-O TERMINAL clause were specified.

INPUT Phrase

INPUT [TERMINAL]

cd-name-1 must reference an input CD when the INPUT phrase is specified.

When the INPUT phrase with the optional word TERMINAL is specified, the logical paths between the source and all of its associated queues and subqueues are deactivated. Only the contents of the data item referenced by *data-name-7* (SYMBOLIC SOURCE) of the area referenced by *cd-name-1* are meaningful.

When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the enabled sources associated with the queues and subqueues specified by the contents of *data-name-1* (SYMBOLIC QUEUE) through *data-name-4* (SYMBOLIC SUB-QUEUE-3) of the area referenced by *cd-name-1* are deactivated.

I-O TERMINAL Phrase

I-O TERMINAL

cd-name-1 must reference an input-output CD when the I-O TERMINAL phrase is specified.

When the I-O TERMINAL phrase is specified, the logical path between the source and the program is deactivated. The source is defined by the contents of the data item referenced by *data-name-3* (SYMBOLIC TERMINAL) of the area referenced by *cd-name-1*.

OUTPUT Phrase

OUTPUT

cd-name-1 must reference an output CD when the OUTPUT phrase is specified.

When the OUTPUT phrase is specified, 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-1*, are deactivated.

TERMINAL Phrase

TERMINAL

cd-name-1 must reference either an input or an input-output CD. If *cd-name-1* refers to an input CD, the DISABLE statement is treated as if it specified the INPUT TERMINAL phrase; if *cd-name-1* refers to an I-O CD, the DISABLE statement is treated as if it specified the I-O TERMINAL phrase.

WITH KEY Phrase

WITH KEY { *identifier-1* / *literal-1* }

In the WITH KEY phrase, *literal-1* or the contents of the data item referenced by *identifier-1* are compared with a password built into the system. The DISABLE statement is honored only if *literal-1* or the contents of the data item referenced by *identifier-1* match the system password. When literal 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-1* is updated.

If the WITH KEY phrase is omitted, the DISABLE statement is honored only if a password is not required by the system.

DISABLE Statement Example

```
DISABLE INPUT INPUT-COM.

DISABLE OUTPUT COM-LINE-1 WITH KEY COM-PASSWORD.
```

DISPLAY . . . UPON Statement

The DISPLAY . . . UPON statement causes individual data items to be displayed on an appropriate hardware device.

Format 1: Display Upon System-Name

```
DISPLAY { identifier-1 } ...  [ UPON { mnemonic-name-3       } ]
        { literal-1    }              { low-volume-I-O-name-1 }

    [ WITH NO ADVANCING ]
```

The DISPLAY statement transfers the contents of each sending operand, *identifier-1* or *literal-1* to the hardware device in the order listed.

In a Format 1 DISPLAY statement, the contents of the data item referred to by *identifier-1* or the value of *literal-1* is transmitted to the standard output device. The presence of the UPON phrase may affect which output device is used. If *mnemonic-name-3* is used in the UPON phrase, it must have been defined in the SPECIAL-NAMES paragraph of the Environment Division with the *low-volume-I-O-name-1* IS *mnemonic-name-3* clause. The associated *low-volume-I-O-name-1* must be CONSOLE or SYSOUT.

The size of a data transfer is determined at program execution time; see the *RM/COBOL User's Guide* for details. If the size of the data item being transferred is not the same as that determined, one of the following applies:

1. If the size of the data item being transferred exceeds the determined size, the data beginning with the leftmost character is displayed aligned to the left on the terminal screen for a length of the determined size, and then this rule is reapplied to the remaining characters to the right until all the data has been transferred.

2. If the size of the data item being transferred is less than the determined size, the transferred data is displayed aligned to the left on the terminal screen.

When the DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes of the operands, and the values of the operands are transferred in the sequence in which the operands are encountered without modifying the positioning of the cursor between the successive operands.

If the WITH NO ADVANCING phrase is not specified, the positioning of the standard output device is reset to the leftmost position of the next line following the transfer of the last operand of the DISPLAY statement.

If the WITH NO ADVANCING phrase is specified, the standard output device remains positioned at the character position immediately following the last character of the last operand displayed.

DISPLAY . . . UPON CONSOLE is treated as if CONSOLE IS CONSOLE was specified in the SPECIAL-NAMES paragraph if CONSOLE has not been otherwise defined.

DISPLAY . . . UPON SYSOUT is treated as if SYSOUT IS SYSOUT was specified in the SPECIAL-NAMES paragraph if SYSOUT has not been otherwise defined.

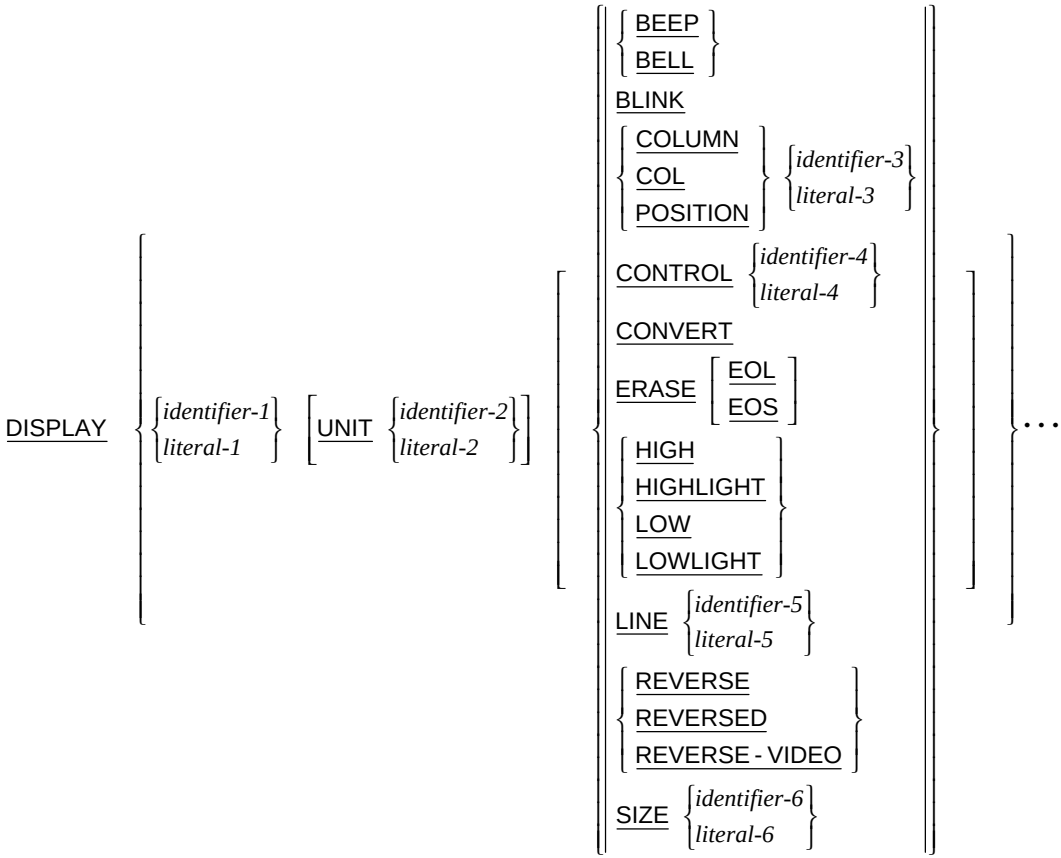
DISPLAY . . . UPON Statement Examples

```
DISPLAY "[" PROMPT-STRING "] " UPON SYSTEM-OUTPUT
  WITH NO ADVANCING.

DISPLAY OPERATOR-MESSAGE UPON CONSOLE.
```

DISPLAY Statement (Terminal I-O)

A terminal I-O DISPLAY statement causes individual data items to be displayed on the terminal screen. DISPLAY statement phrases allow the specification of the position, form, and format of the displayed data.

Format 2: Display Terminal I-O



The DISPLAY statement transfers the contents of each sending operand, *identifier-1* or *literal-1* to the terminal screen in the order listed.

If a figurative constant is specified as one of the sending operands, only a single occurrence of the figurative constant is displayed, except as specified in the rules for the SIZE phrase on page 6-67.

identifier-2, *identifier-3*, *identifier-5* and *identifier-6* must be described as integer numeric data items. *literal-2*, *literal-3*, *literal-5* and *literal-6* must be nonnegative integer numeric literals.

identifier-4 must be a nonnumeric data item. *literal-4* must be a nonnumeric literal.

Several terms are used to describe the detailed function of each phrase in a Format 2 DISPLAY statement:

1. The term "output field" describes a conceptual data item containing the data transmitted to the terminal and displayed on the terminal screen. The size of this data item is determined according to the rules described below (see the discussion of the SIZE phrase that begins on page 6-67), and the type of this data item is alphanumeric.

2. The term "sending item" is synonymous with the data item *identifier-1* or *literal-1*.

3. The term "screen field" applies to the physical field presented on the screen itself.

Table 6-4 shows the relationships of the various Format 2 DISPLAY statement phrases to the characteristics of the output field and the screen field subject to control by the program.

Table 6-4: DISPLAY Statement Phrases and Output and Screen Fields

Characteristic	Phrases
Screen field position	LINE, POSITION, ERASE, SIZE, UNIT, CONTROL
Screen field size	*identifier-1*, CONVERT, SIZE, CONTROL
Visual attributes	ERASE, HIGH, LOW, BLINK, REVERSE, CONTROL
Audio attribute	BEEP, CONTROL
Output conversion	CONVERT, CONTROL

Note that the CONTROL phrase may be used in many instances to allow dynamic (that is, runtime as opposed to compile time) specification of characteristics.

Features that require support of the host operating system or terminal hardware may not be supported in all circumstances. Any features that are not supported compile correctly but are ignored at runtime. See the *RM/COBOL User's Guide* for each implementation environment in order to obtain specific details. Also note that some phrases may require that character positions on the screen between fields be reserved for attribute characters

(typically, to support the HIGH, LOW, BLINK, REVERSE, ERASE EOL and ERASE EOS phrases). It is the programmer's responsibility to allow for attribute characters by not juxtaposing fields that may require them.

The phrases following a sending operand apply only to that operand. When the DISPLAY statement contains multiple sending operands and any of the phrases are omitted for a particular operand, the defaults described below for that phrase are applied to that operand.

BEEP Phrase

```
{ BEEP }
{ BELL }
```

BELL is a synonym for BEEP.

The presence of the BEEP phrase in a DISPLAY statement causes the audio alarm signal to occur prior to the display of the data. If the BEEP phrase is omitted, no signal is given.

BLINK Phrase

```
BLINK
```

The presence of the BLINK phrase causes the data to be displayed in a blinking mode. If the BLINK phrase is not specified, the data is displayed in a nonblinking mode.

CONTROL Phrase

```
CONTROL { identifier-4 }
        { literal-4    }
```

The value of *identifier-4* or *literal-4* in the CONTROL phrase is used to specify a dynamic option list. The value must be a character-string consisting of a series of keywords delimited by commas; some keywords allow assignment of a value by following the keyword with an equal sign and the value. Blanks are ignored in the character-string. Lowercase letters are treated as uppercase letters within keywords. Keywords specified override corresponding static options specified as phrases for the same sending item. Keywords may be specified in any order. Keywords which specify options that do not apply to the statement are ignored.

The keywords that affect a DISPLAY statement are BEEP, BLINK, CONVERT, ERASE, ERASE EOL, ERASE EOS, HIGH, LOW, NO BEEP, NO BLINK, NO CONVERT, NO ERASE, NO REVERSE, NO UNDERLINE, REVERSE and UNDERLINE. The

meanings of these keywords when they appear in the value of the CONTROL phrase operand are the same as the corresponding phrases which may be written as static options of the DISPLAY statement, with the addition of the negative forms to allow suppression of statically declared options. The keyword UNDERLINE is an exception. It is not recognized as a static option, but it may be used in the value of the CONTROL phrase operand. When it is used there it causes the field on the screen to be shown in underline mode, provided the terminal supports that mode. Additional keywords may be supported in environments that have device-dependent functions (for example, color control); see the *RM/COBOL User's Guide* for the specific implementation.

The keywords are grouped by function such that only the rightmost appearance in the control value of a keyword from a functional group actually affects the screen field. The groupings are as follows:

1. Erasure: ERASE, ERASE EOL, ERASE EOS, NO ERASE
2. Alarm: BEEP, NO BEEP
3. Intensity: HIGH, LOW, OFF
4. Blinking: BLINK, NO BLINK
5. Video: REVERSE, NO REVERSE
6. Output data conversion: CONVERT, NO CONVERT
7. Underscoring: UNDERLINE

CONVERT Phrase

CONVERT

The presence of the CONVERT phrase causes the contents of the sending item to be converted before being moved to the output field and displayed.

If the sending item is numeric or numeric edited and CONVERT is specified, the value of the sending item is converted from its internal form into display digits which are moved to the output field with leading zero digits removed. The display digits are left justified in the output field, with a leading, separate minus sign provided if the value is negative and an explicit decimal point provided if the sending item is noninteger. The representation of this explicit decimal point is a period, except that, if the DECIMAL-POINT IS COMMA clause is specified in the source program, a comma is used instead. Unused character positions to the right of the converted number in the output field are space filled. If the SIZE phrase specifies a value too small for the converted number, the string resulting from the conversion is truncated on the right.

If the sending item is nonnumeric, or if the CONVERT phrase is not specified, the sending item is treated as an alphanumeric item and the contents of the sending item are moved to the output field according to the rules of a simple alphanumeric move (that is, left justified, with space fill to the right).

ERASE Phrase

The presence of the ERASE phrase without either of the reserved words EOL or EOS causes the entire screen of the terminal to be erased. The current line and current position are set to 1.

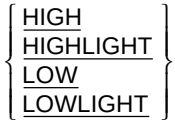
The presence of the ERASE EOL phrase causes the portion of the line containing the leftmost character of the screen field to be erased from the leftmost character of the screen field to the rightmost character of that line.

The presence of the ERASE EOS phrase causes the portion of the screen to be erased from the leftmost character of the screen field to the rightmost character of the bottom line of the screen.

In all three cases above, erasure occurs before any data is displayed in the screen field.

When the ERASE phrase is not specified, no erasure occurs before displaying the data. The displayed data will replace any previous contents of the screen field and the remainder of the screen will be undisturbed.

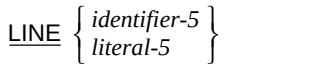
HIGH and LOW Phrases

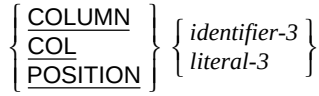


HIGHLIGHT is a synonym for HIGH. LOWLIGHT is a synonym for LOW.

The presence of the HIGH or LOW phrase causes the data to be displayed at the specified intensity. When HIGH or LOW is not specified, the default intensity is HIGH.

LINE and POSITION Phrases





COLUMN and COL are synonyms for POSITION.

The screen field is positioned on the terminal screen by specifying the line and position (that is, the character position within the line) of the leftmost character of the screen field. The top line of the terminal screen is line 1, the line below line 1 is line 2, and so forth. The rightmost character position of a line is immediately followed by the leftmost character position (position 1) of the line below; a screen field may overlap line boundaries on the terminal screen. The leftmost character of the screen field refers to the leftmost character position of that portion of the screen field that is on the topmost line containing a portion of the screen field. Similarly, the rightmost character position of the screen field refers to the rightmost character position of that portion of the screen field that is on the bottommost line containing a portion of the screen field.

The current line and current position prior to the DISPLAY operation for each *identifier-1* may affect the position of the screen field as described in the rules below. At the beginning of a run unit, the current line is the last (bottommost) line and the current position is the leftmost (position 1) of that line. The current line and current position are changed by each Format 3 ACCEPT and Format 2 DISPLAY operation to be the line and position of the character immediately succeeding the rightmost character of the screen field. If the ERASE phrase (without EOL or EOS) is specified for the same *identifier-1*, the current line and current position are both set to 1.

The value of *identifier-5* or *literal-5* in the LINE phrase specifies the line value for the leftmost character of the screen field. The value of *identifier-3 or literal-3* in the POSITION phrase specifies the position value for the leftmost character of the screen field.

Determining Line and Position

If the POSITION phrase is omitted, the position value is set to 1 for the first *identifier-1* of a Format 2 DISPLAY statement; this value is also set to 1 if a UNIT phrase is specified for the same *identifier-1*. It is set to zero in all other cases.

If the line value is zero, or if the LINE phrase is omitted, the line value is set according to the following rules:

1. If an ERASE phrase (without EOL or EOS) is specified for the same *identifier-1*, the line value is set to 1.

2. If the position value is not equal to zero, the line value is set to the current line plus 1.

3. If the position value is equal to zero, the line value is set to the current line.

If the position value is greater than the maximum number of characters within a line, the position value is reduced by the maximum number of characters within a line and the line value is incremented by 1. This process is repeated until the position value is not greater than the maximum number of characters within a line.

If the position value is equal to zero, the position value is set to the current position.

If the line value exceeds the number of lines on the screen, the contents of the screen are scrolled up one line and the line value is set to the number of lines on the screen.

If the line of the rightmost character of the screen field exceeds the number of lines on the screen, the contents of the screen are scrolled up the amount of the excess and the line value is reduced by the amount of the excess.

The resulting line value and position value specify the position of the leftmost character of the screen field.

REVERSE Phrase

REVERSED and REVERSE-VIDEO are synonyms for REVERSE.

The presence of the REVERSE phrase causes the data to be displayed in a reverse video mode. If the REVERSE phrase is not specified, the data is displayed in normal video mode.

SIZE Phrase

SIZE { *identifier-6* / *literal-6* }

The value of *identifier-6* or *literal-6* in the SIZE phrase specifies the size of the screen field and the output field.

If the SIZE phrase is not present or a value of zero is specified, the size of *identifier-1* or *literal-1* is used. If *identifier-1* or *literal-1* is numeric or numeric edited and the CONVERT phrase is specified for the same *identifier-1* or *literal-1*, the size is considered to be the number of digits (9's and P's) defined in the PICTURE character-string or literal plus one if the item is signed and plus one if the item is noninteger.

If *literal-1* is a figurative constant and the SIZE phrase is specified, then the figurative constant is repeated to match the specified size before being displayed.

UNIT Phrase

UNIT { *identifier-2* / *literal-2* }

The UNIT phrase, if specified, must be written first. The other phrases may be written in any order.

The value of *identifier-2* or *literal-2* in the UNIT phrase specifies the terminal upon which the data is to be displayed. If the UNIT phrase is omitted, the terminal that started the run unit is used.

The UNIT phrase may be ignored by some runtime implementations except in its effect on the default value of the POSITION phrase (described previously). This situation will occur in all systems that do not allow the use of terminals other than the one associated with the run unit execution.

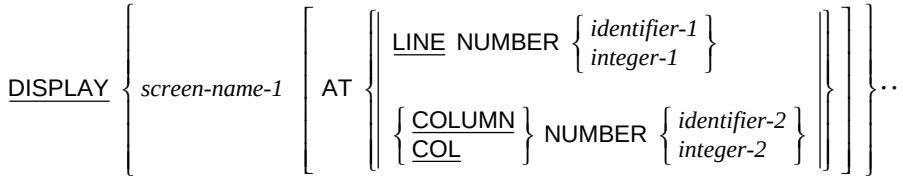
DISPLAY Statement (Terminal I-O) Examples

```
DISPLAY "Flight arriving at gate:", LINE FLT-LN,
  POSITION 1, ERASE; GATE-NUMBER, HIGH, BLINK.


DISPLAY "Enter job code: " LINE 12 COLUMN 5.


DISPLAY MENU-HEADER LINE 1 ERASE HIGH.


DISPLAY ZEROES SIZE 5.  *> displays "00000"


DISPLAY QUOTE. *> displays """" (one quote character)


DISPLAY REPORT-LINE CONTROL "HIGH, ERASE EOL".


DISPLAY display-data (ix),
  LINE display-line (ix),
  COL  display-column (ix),
  SIZE display-size (ix),
  CONTROL display-control (ix).
```

DISPLAY Screen-Name Statement

The DISPLAY Screen-Name statement moves data onto the terminal screen from literals or from data items defined in the Data Division. The organization, placement and visual attributes of the fields on the screen are defined in the Screen Section of the Data Division.

Format 3: Display Screen-Name



COL is a synonym for COLUMN.

A DISPLAY statement that specifies multiple screen names is equivalent to a series of DISPLAY statements, one for each of the specified screen names.

Each *screen-name-1* must be defined as an elementary or group entry in the Screen Section of the Data Division. If *screen-name-1* is an elementary item it is treated as if it were a group consisting of the single elementary item to which it refers. *identifier-1* and *identifier-2*, when used, must refer to elementary numeric integer data items.

If the LINE phrase is specified, the value of *integer-1* or the current value of the data item referred to by *identifier-1* is used as an increment to each of the explicit or implicit LINE specifications within *screen-name*, thus shifting the screen downward the specified number of lines.

A similar rule applies if the COLUMN phrase is specified: the value of *integer-2* or the current value of the data item referred to by *identifier-2* is used as an increment to each of the explicit or implicit COLUMN specifications within *screen-name-1*, thus shifting the screen to the right the specified number of columns.

Each elementary item subordinate to *screen-name-1* is acted on in response to a DISPLAY statement. Areas of the screen not specifically changed by fields within *screen-name-1* remain unchanged. All the attributes meaningful for output operations are effective. This excludes AUTO, FULL, REQUIRED and SECURE. For fields defined with a VALUE clause, the literal is moved to the screen field. For fields defined with a PICTURE clause that has a FROM or USING option, the value of the associated item is moved to the screen field item and to the retained value. For fields defined with a

PICTURE clause that has the TO option and no FROM option, the screen field and the retained value are filled with underline characters.

Numeric data items are always displayed with output conversion. (See the discussion of the CONVERT phrase on page 6-63 for an explanation of output conversion.)

The appearance of the screen is undefined and unpredictable when LINE or COLUMN values are specified such that screen fields extend beyond the boundaries of the physical screen, either horizontally or vertically.

DISPLAY Screen-Name Statement Examples

```
DISPLAY INVOICE-FORM LINE 10 COLUMN 5.

DISPLAY EMPLOYEE-RECORD AT LINE 9.

DISPLAY EOB-SCREEN AT COL EOB-COL LINE EOB-LINE.
```

DIVIDE Statement

The DIVIDE statement divides one numeric data item into another and stores the quotient and remainder.

Format 1: Divide…Into

```
DIVIDE { identifier-1 } INTO { identifier-2 [ ROUNDED ] }...
       { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

Format 2: Divide…Into…Giving

```
DIVIDE { identifier-1 } INTO { identifier-2 }
       { literal-1    }      { literal-2    }
    GIVING { identifier-3 [ ROUNDED ] }...
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

Format 3: Divide…By…Giving

```
DIVIDE { identifier-2 } BY { identifier-1 }
       { literal-2    }    { literal-1    }
    GIVING { identifier-3 [ ROUNDED ] }...
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

Format 4: Divide…Into…Giving…Remainder

```
DIVIDE { identifier-1 } INTO { identifier-2 }
       { literal-1    }      { literal-2    }

    GIVING identifier-3 [ ROUNDED ] REMAINDER identifier-4

    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ]

    [ END-DIVIDE ]
```

Format 5: Divide…By…Giving…Remainder

```
DIVIDE { identifier-2 } BY { identifier-1 }
       { literal-2    }    { literal-1    }

    GIVING identifier-3 [ ROUNDED ] REMAINDER identifier-4

    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ]

    [ END-DIVIDE ]
```

In Format 1, the value of *identifier-1* or *literal-1* is divided into the value of each *identifier-2*. The value of each dividend (*identifier-2*) is replaced by this quotient.

In Format 2, the value of *identifier-1* or *literal-1* is divided into the value of *identifier-2* or *literal-2* and the result is stored in each *identifier-3*.

In Format 3, the value of *identifier-2* or *literal-2* is divided by the value of *identifier-1* or *literal-1* and the result is stored in each *identifier-3*.

In Format 4, the value of *identifier-1* or *literal-1* is divided into the value of *identifier-2* or *literal-2*; the result is stored in *identifier-3*, and the remainder is stored in *identifier-4*.

In Format 5, the value of *identifier-2* or *literal-2* is divided by the value of *identifier-1* or *literal-1*; the result is stored in *identifier-3*, and the remainder is stored in *identifier-4*.

Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING phrase may refer to either an elementary numeric item or an elementary numeric edited item.

Each literal must be a numeric literal.

Additional rules and explanations regarding features of the DIVIDE statement that are common to other arithmetic statements can be found in the discussion of common rules that begins on page 5-16. See in particular the discussions of the ROUNDED phrase, the size error condition, overlapping operands, modes of operation, composite size, and incompatible data.

REMAINDER Phrase

GIVING *identifier-3* [ROUNDED] REMAINDER *identifier-4*

Formats 4 and 5 are used when a remainder from the division operation is desired, namely *identifier-4*. The remainder 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 that contains the unedited quotient. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field which contains the quotient of the DIVIDE statement, truncated rather than rounded. The intermediate field used in these calculations has the same number of digit positions and the same scale as *identifier-3*.

In Formats 4 and 5, the accuracy of the REMAINDER data item (*identifier-4*) is defined by the calculation described above.

Appropriate decimal alignment and truncation (not rounding) is performed for the content of the data item referenced by *identifier-4*, as needed. When the composite of the quotient and dividend operands contains more than 19 digits, the accuracy of the REMAINDER data item may be greater than that obtainable by the use of a COMPUTE statement which duplicates the calculation described above.

When the ON SIZE ERROR phrase is used in Formats 4 and 5, the following rules pertain:

1. If the size error condition occurs on the quotient, no remainder calculation is meaningful. Thus, the contents of the data items referenced by both *identifier-3* and *identifier-4* remain unchanged.

2. If the size error condition occurs on the remainder, the contents of the data item referenced by *identifier-4* remain unchanged.

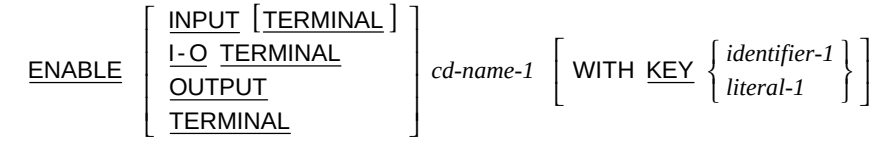
It is the user's responsibility to determine which situation has actually occurred.

DIVIDE Statement Examples

```
DIVIDE 10 INTO TOTAL-WORK-LOAD. *>  10 FTEs


DIVIDE 6 INTO TOTAL-WORK-LOAD   *>   6 FTEs
  GIVING AVERAGE-WORK-LOAD.


DIVIDE TOTAL-WORK-LOAD BY 2.5   *> 2.5 FTEs
  GIVING AVERAGE-WORK-LOAD
ON SIZE ERROR PERFORM OVERFLOW-ROUTINE
END-DIVIDE.


DIVIDE DIVISOR-1 INTO DIVIDEND-1
  GIVING QUOTIENT-1 ROUNDED
  REMAINDER REMAINDER-1.


DIVIDE DIVIDEND-1 BY DIVISOR-1
  GIVING QUOTIENT-1
  REMAINDER REMAINDER-1
ON SIZE ERROR  MOVE "E" TO SIZE-ERROR-FLAG
END-DIVIDE.
```

ENABLE Statement

The ENABLE statement notifies the Message Control System (MCS) to allow data transfer between specified output queues and destinations for output, between specified sources and input queues for input or between the program and one specified source or destination for input-output.



cd-name-1 is defined below for each phrase.

identifier-1 must refer to a data item of category alphanumeric.

literal-1 must be a nonnumeric literal.

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 a means external to this program, the ENABLE statement is not required in this program. No action is taken when an ENABLE statement is executed which specifies a source or destination that is already connected, except that the value in the status key data item indicates this condition. The logical path for the transfer of data between the object programs and the MCS is not affected by the ENABLE statement.

An ENABLE statement that lacks an INPUT, OUTPUT, I-O or TERMINAL keyword is treated according to the format of the description of the cd-name:

- An ENABLE statement that refers to an INPUT cd-name and does not specify the INPUT keyword is treated as if the INPUT phrase without the keyword TERMINAL were specified.
- An ENABLE statement that refers to an OUTPUT cd-name and does not specify the OUTPUT keyword is treated as if the OUTPUT phrase were specified.
- An ENABLE statement that refers to an I-O cd-name and does not specify the I-O keyword is treated as if the I-O TERMINAL phrase were specified.

INPUT Phrase

INPUT [TERMINAL]

cd-name-1 must reference an input CD when the INPUT phrase is specified.

When the INPUT phrase with the optional word TERMINAL is specified, the logical paths between the source and all of its associated queues and subqueues are activated. Only the contents of the data item referenced by *data-name-7* (SYMBOLIC SOURCE) of the area referenced by *cd-name-1* are meaningful to the MCS.

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 SUB-QUEUE-3) of the area referenced by *cd-name-1* are activated.

I-O TERMINAL Phrase

I-O TERMINAL

cd-name-1 must reference an input-output CD when the I-O TERMINAL phrase is specified.

When the I-O TERMINAL phrase is specified, the logical path between the source and the program is activated. The source is defined by the contents of the data item referenced by *data-name-3* (SYMBOLIC TERMINAL) of the area referenced by *cd-name-1*.

OUTPUT Phrase

OUTPUT

cd-name-1 must reference an output CD when the OUTPUT phrase is specified.

When the OUTPUT phrase is specified, 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-1* are activated.

TERMINAL Phrase

TERMINAL

cd-name-1 must reference either an input or an input-output CD. If *cd-name-1* refers to an input CD, the ENABLE statement is treated as if it specified the INPUT TERMINAL phrase; if *cd-name-1* refers to an I-O CD, the ENABLE statement is treated as if it specified the I-O TERMINAL phrase.

WITH KEY Phrase

WITH KEY { *identifier-1* / *literal-1* }

In the WITH KEY phrase, *literal-1* or the contents of the data item referenced by *identifier-1* are compared with a password built into the system. The ENABLE statement is 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-1* is updated.

If the WITH KEY phrase is omitted, the ENABLE statement is honored only if a password is not required by the system.

ENABLE Statement Examples

```
ENABLE INPUT TERMINAL COM-PORT.

ENABLE OUTPUT COM-LINE-1 WITH KEY COM-PASSWORD.
```

ENTER Statement

The ENTER statement provides a means of allowing the use of more than one language in the same program. In RM/COBOL, no other source language is allowed in the source program.

ENTER *language-name-1* [*routine-name-1*]

language-name-1 may be any COBOL word.

routine-name-1 is a COBOL word and may be referred to only in an ENTER sentence.

The ENTER statement must appear only in a sentence by itself.

The sentence ENTER COBOL must follow the last statement of the other language in order to indicate to the compiler where a return to COBOL source language takes place. It must be followed by a separator period.

The statements of the other language are executed in the object program as if they had been compiled into the object program following the ENTER statement.

No other languages may appear in a COBOL source program following an ENTER statement.

routine-name-1 indicates the portion of other-language coding to be executed at this point in the procedure sequence when the entered language cannot be written in-line. If the other language statements are written in-line, *routine-name-1* is not used.

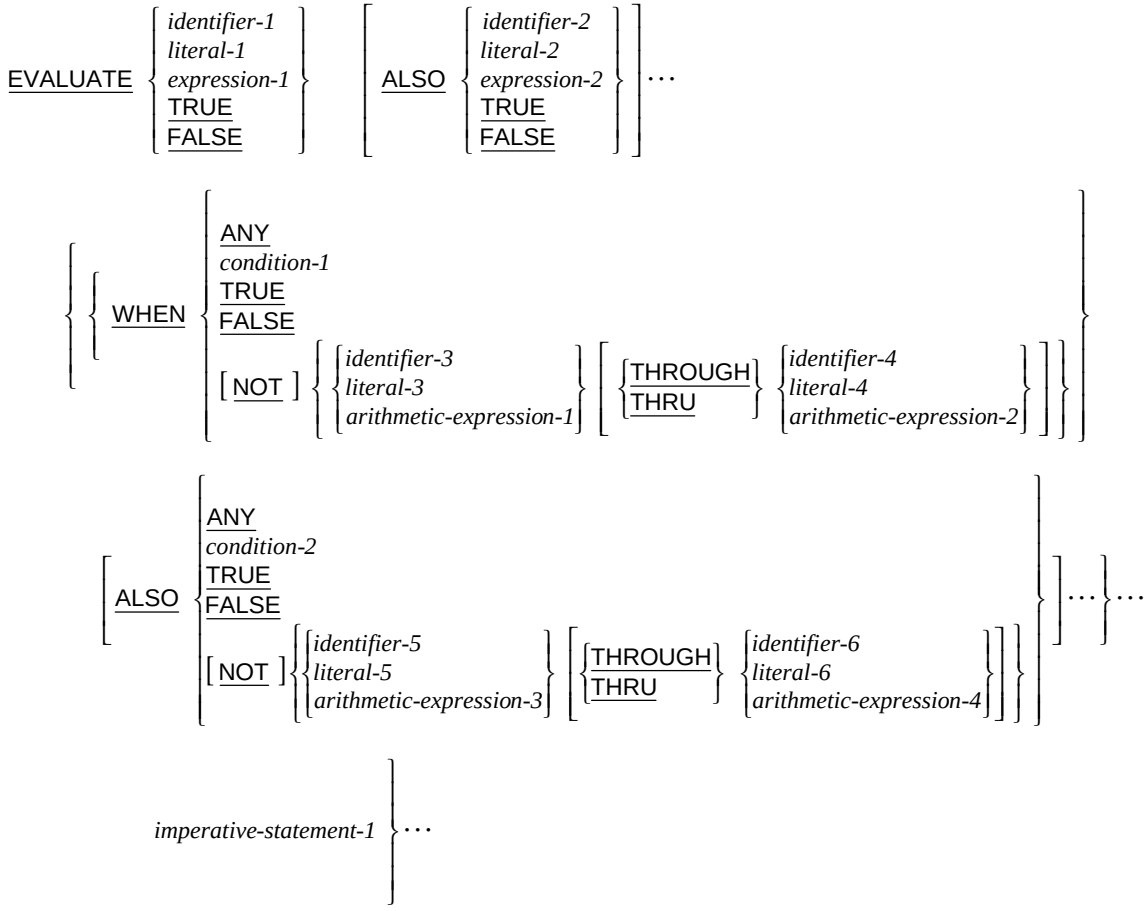
The ENTER statement is accepted as commentary for compatibility with other COBOL implementations. The CALL statement may be used to execute object programs from other language processors.

ENTER Statement Examples

```
ENTER LINKAGE.
CALL "SUBROUTINE" USING ARGUMENT-GROUP.
ENTER COBOL.


ENTER FORTRAN SUBROUTINE-1.
```

EVALUATE Statement

The EVALUATE statement describes a multibranch, multijoin structure. It can cause multiple conditions to be evaluated. The subsequent action of the object program depends on the results of these evaluations.



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.

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.

The words THROUGH and THRU are synonymous.

Two operands connected by a THROUGH phrase must be of the same class. The two operands thus connected constitute a single selection object.

The number of selection objects within each set of selection objects must be equal to the number of selection subjects.

Each selection object within a set of selection objects must correspond to the selection subject having the same ordinal position within the set of selection subjects according to the following rules:

1. 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.

2. *condition* or the word TRUE or FALSE appearing as a selection object must correspond to a conditional expression or the word TRUE or FALSE in the set of selection subjects.

3. The word ANY may correspond to a selection subject of any type.

General Rules

The general rules that apply to the EVALUATE statement are as follows:

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 phrase, 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 phrase, are assigned the value and class of the specified literal. If *literal-3* is the figurative constant ZERO, it is assigned the class of the corresponding selection subject.

c. Any selection subject in which *arith-expr-1*, *arith-expr-2* is specified as an arithmetic expression and any selection object, without either the NOT or the THROUGH phrase, in which *arith-expr-3*, *arith-expr-5* is specified, are assigned a numeric value according to the rules for evaluating an arithmetic expression.

d. Any selection subject in which *condition-1*, *condition-2* is specified as a conditional expression and any selection object in which *condition-3*, *condition-4* is specified, are assigned a truth value according to the rules for evaluating conditional expressions.

e. Any selection subject or any selection object specified by the word 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 includes all permissible values of the selection subject that are greater than or equal to the first operand and less than or equal to the second operand according to the rules for comparison.

h. If the NOT phrase is specified for a selection object, the values assigned to that item are all permissible values of the selection subject not equal to the value, or not included in the range of values, that 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 phrase satisfies 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 having the same ordinal position within the set of selection subjects.

b. One of the following conditions must be satisfied if the comparison is to be satisfied:

   (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.

   (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 always satisfied regardless of the value of the selection subject.

c. If the above comparison is satisfied for every selection object within the set of selection objects being compared, the WHEN phrase containing that set of selection objects is selected as the one satisfying the set of selection subjects.

d. 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.

e. This procedure is repeated for subsequent sets of selection objects, in the order of their appearance in the source program, until either a WHEN phrase satisfying the set of selection subjects is selected or until all sets of selection objects are exhausted.

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 scope of execution of the EVALUATE statement is terminated when execution reaches the end of *imperative-statement-1* of the selected WHEN phrase or the end of *imperative-statement-2*, or when no WHEN phrase is selected and no WHEN OTHER phrase is specified.

EVALUATE Statement Examples

```
EVALUATE OPERATION-TYPE
WHEN TYPE-UPDATE PERFORM UPDATE-IT
WHEN TYPE-DELETE PERFORM DELETE-IT
WHEN TYPE-INSERT PERFORM INSERT-IT
WHEN OTHER PERFORM BAD-OPERATION-TYPE
END-EVALUATE.
```

```
EVALUATE DAY-VALUE ALSO LEVEL-VALUE
WHEN 1 ALSO ANY        PERFORM MONDAY-PROCESSING
WHEN 2 THRU 4 ALSO "SUMMARY"
  PERFORM MIDWEEK-PROCESSING
WHEN 2 ALSO "DETAILED" PERFORM TUESDAY-PROCESSING
WHEN 3 ALSO "DETAILED" PERFORM WEDNESDAY-PROCESSING
WHEN 4 ALSO "DETAILED" PERFORM THURSDAY-PROCESSING
WHEN 5 ALSO ANY        PERFORM FRIDAY-PROCESSING
WHEN 6 ALSO ANY
WHEN 7 ALSO ANY        PERFORM WEEKEND-PROCESSING
WHEN OTHER             PERFORM BAD-DAY-OR-LEVEL
END-EVALUATE.
```

```
EVALUATE TRUE
WHEN ANNUALLY AND YEAR-END
  PERFORM ANNUAL-UPDATE
WHEN QUARTERLY AND QUARTER-END
  PERFORM QUARTER-UPDATE
WHEN MONTHLY AND MONTH-END
  PERFORM MONTH-UPDATE
END-EVALUATE.
```

EXIT Statement

The EXIT statement provides a common end point for a series of procedures. The EXIT PROGRAM statement marks the logical end of a called program. The EXIT PERFORM statement provides a means of exiting an in-line PERFORM (with or without returning to any specified test). The EXIT PARAGRAPH or EXIT SECTION statements provide a means of exiting a structured procedure without executing any of the following statements within a procedure.

Format 1: Exit Paragraph

```
EXIT
```

Format 2: Exit Program

```
EXIT PROGRAM
```

Format 3: Exit In-Line Perform

```
EXIT PERFORM [ CYCLE ]
```

Format 4: Exit Paragraph or Section

```
EXIT { PARAGRAPH }
     { SECTION   }
```

The Format 1 EXIT statement must appear in a sentence by itself, and that sentence must be the only sentence in the paragraph.

If a Format 2, Format 3, or Format 4 EXIT statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

The Format 2 EXIT PROGRAM statement must not appear in a declarative procedure in which the GLOBAL phrase is specified.

The Format 1 EXIT statement allows 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. The Format 1 EXIT statement, together with its paragraph-name, is equivalent to an empty paragraph.

If the Format 2 EXIT PROGRAM statement is executed in a program which is not under the control of a calling program, the EXIT PROGRAM statement causes execution of the program to continue with the next executable statement.

The execution of a Format 2 EXIT PROGRAM statement in a called program which does not possess the initial attribute causes execution to continue with the next executable statement following the CALL statement in the calling program. The program state of the calling program is not altered and is identical to that which existed at the time it executed the CALL statement except that the contents of data items and the contents of data files shared between the calling and called program may have been changed. The program state of the called program is not altered except that the ends of the ranges of all PERFORM statements executed by that called program are considered to have been reached.

Besides the actions specified in the preceding paragraph, the execution of an EXIT PROGRAM statement in a called program which possesses the initial attribute is equivalent also to executing a CANCEL statement referencing that program.

The Format 3 EXIT PERFORM statement may be specified only in an in-line PERFORM statement.

The execution of a Format 3 EXIT PERFORM statement without the CYCLE phrase causes control to be passed to an implicit CONTINUE statement immediately following the END-PERFORM phrase that matches the most closely preceding, and as yet unterminated, in-line PERFORM statement.

The execution of a Format 3 EXIT PERFORM statement with the CYCLE phrase causes control to be passed to an implicit CONTINUE statement immediately preceding the END-PERFORM phrase that matches the most closely preceding, and as yet unterminated, in-line PERFORM statement.

The Format 4 EXIT statement with the PARAGRAPH phrase may be specified only in a paragraph.

The execution of a Format 4 EXIT statement with the PARAGRAPH phrase causes control to be passed to an implicit CONTINUE statement immediately following the last statement in the current paragraph.

The Format 4 EXIT statement with the SECTION phrase may be specified only in a section.

The execution of a Format 4 EXIT statement with the SECTION phrase causes control to be passed to an implicit CONTINUE statement within an implicit paragraph immediately following the last statement in the current section.

Exit Statement Examples

```
        PERFORM WEEKEND-PROC THRU WEEKEND-PROC-EXIT.
           ⋮
    WEEKEND-PROC.
           ⋮
    WEEKEND-PROC-CONT.
           ⋮
    WEEKEND-PROC-EXIT.
        EXIT.


        IF RECORD-TYPE NOT = MY-RECORD-TYPE
        THEN
            MOVE 4096 TO RETURN-CODE
            EXIT PROGRAM
        END-IF.
```

GOBACK Statement

The GOBACK statement specifies the logical end of a called program.

GOBACK

The GOBACK statement is equivalent to the sequence:

```
EXIT PROGRAM.
STOP RUN.
```

The GOBACK statement must appear as the only statement, or as the last of a series of imperative statements, in a sentence.

The GOBACK statement must not appear in a declarative procedure in which the GLOBAL phrase is specified.

If control reaches a GOBACK statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement. For details, see the discussion of the EXIT PROGRAM statement that begins on page 6-84.

If no CALL statement is active and the GOBACK statement is executed in the main program, control returns to the invoker (which may be the operating system and thus cause the end of the run unit).

GOBACK Statement Examples

```
GOBACK.


IF RECORD-TYPE NOT = MY-RECORD-TYPE
THEN
  MOVE 4096 TO RETURN-CODE
  GOBACK
END-IF.
```

GO TO Statement

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

Format 1: Go To (Alterable)

GO TO [*procedure-name-1*]

Format 2: Go To (Non-Alterable)

GO TO *procedure-name-1*

Format 3: Go To…Depending On

GO TO { *procedure-name-1* }… DEPENDING ON *identifier-1*

A Format 1 GO TO statement can only appear in a single statement paragraph and can be altered with an ALTER statement.

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.

If *procedure-name-1* is not specified in Format 1, an ALTER statement, referring to the paragraph containing this GO TO statement, must be executed prior to the execution of this GO TO statement; otherwise, the run unit is terminated with an error message when the GO TO statement is executed.

When a Format 1 or 2 GO TO statement 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.

If a Format 2 GO TO statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

DEPENDING ON Phrase

DEPENDING ON *identifier-1*

When a Format 3 GO TO statement is executed, control is transferred to *procedure-name-1* depending on the value of *identifier-1* being 1, 2, . . ., *n*. If the value of *identifier-1* is anything other than the positive or unsigned integers 1, 2, . . ., *n*, no transfer occurs and control passes to the next statement in the normal sequence for execution.

identifier-1 must refer to a numeric integer elementary data item.

GO TO Statement Example

```
       IF STATE-1-UP
         ALTER STATE-1-SWITCH TO STATE-1-UP-PROC
       ELSE
         ALTER STATE-1-SWITCH TO STATE-1-DOWN-PROC.
           ⋮
   STATE-1-SWITCH.
       GO TO.
           ⋮
   STATE-1-UP-PROC.
           ⋮
   STATE-1-DOWN-PROC.
           ⋮


       GO TO STATE-1-EXIT-PROC.


       GO TO CHOICE-1, CHOICE-2, CHOICE-3
         DEPENDING ON USER-PICK.
```

IF Statement

The IF statement causes a specified condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

```
IF condition-1  THEN { statement-1      }
                     { NEXT SENTENCE    }

   [ ELSE { statement-2      } ]
          { NEXT SENTENCE    }

   [ END-IF ]
```

statement-1 and *statement-2* each represent either an imperative statement or a conditional statement optionally preceded by an imperative statement.

The scope of an IF statement is terminated by any of the following:

- An END-IF phrase at the same level of nesting.
- A separator period.
- If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting.
- The next phrase of any statement in which the IF statement is contained.

When an IF statement is executed, the following transfers of control occur:

- If *condition-1* 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.
- If *condition-1* 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.
- If *condition-1* 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; otherwise, upon the completion of *statement-2* 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.

- If *condition-1* is false, and the ELSE NEXT SENTENCE phrase is specified, *statement-1* is ignored, if specified, and control passes to the next executable sentence.

Either *statement-1* or *statement-2* may contain an IF statement. When this occurs, the IF statement is said to be nested.

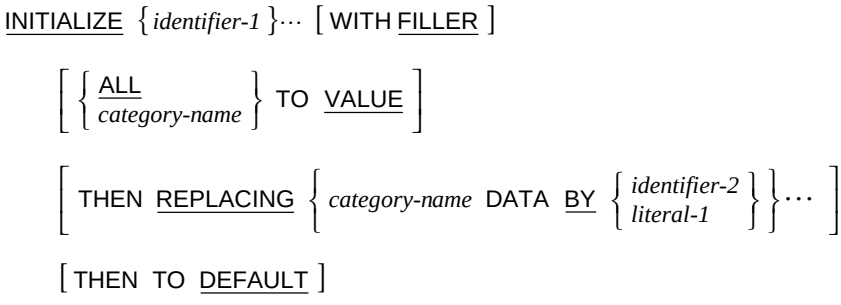
IF statements within IF statements may be considered as paired IF and ELSE and END-IF combinations, proceeding from left to right. Thus, any ELSE or END-IF encountered is considered to apply to the most recent preceding IF that has not been already paired with an ELSE or END-IF.

IF Statement Examples

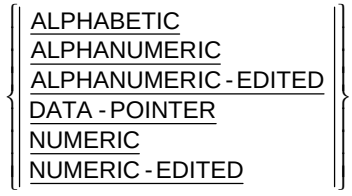
```
IF CHAR-STR IS ALPHABETIC
THEN MOVE CHAR-STR TO ALPHA-STR;
ELSE IF CHAR-STR IS NUMERIC
THEN MOVE CHAR-STR TO NUM;
ELSE NEXT SENTENCE.

IF NUM = OLD-NUM GO TO RE-SET.

IF ALPHA-STR NOT = "TEST"
  ADD 1 TO ERROR-CNT
  IF ERROR-CNT >= 20
    DISPLAY "Excessive errors."
    STOP RUN
  END-IF
ELSE
  PERFORM TEST-PROCEDURE
END-IF.

IF NUM < UPPER-LIMIT, ADD 1 TO NUM.

IF NUM IS LESS THAN UPPER-LIMIT
THEN
  ADD 1 TO NUM
ELSE
  PERFORM RE-SET
END-IF.

IF PRINT-SWITCH-ON PERFORM PRINT-ROUTINE.
```

INITIALIZE Statement

The INITIALIZE statement provides the ability to set selected types of data fields to predetermined values; for example, numeric data to zeroes, alphanumeric data to spaces, or data pointers to NULL.



where *category-name* is:



identifier-1 must be of class alphabetic, alphanumeric, numeric, or data pointer.

For the category data-pointer specified in the REPLACING phrase, a SET statement with *identifier-2* or *literal-1* as the sending operand and an item of the category data-pointer as the receiving operand must be valid.

For each of the other categories specified in the REPLACING phrase, a MOVE statement with *identifier-2* or *literal-1* as the sending item and an item of the specified category as the receiving operand must be valid.

The same category cannot be repeated in a REPLACING phrase.

An index data item may not appear as an operand of an INITIALIZE statement.

The data description entry for the data item referenced by *identifier-1* must not contain a RENAMES clause.

General Rules

The general rules that apply to the INITIALIZE statement are as follows:

1. The data item referenced by *identifier-1* represents the receiving item.

2. If the REPLACING phrase is specified, *literal-1* or the data item referenced by *identifier-2* represents a possible sending item as specified in general rule 6.

3. The keywords in *category-name* correspond to a category of data as defined in the discussion of the PICTURE clause that begins on page 4-34, or for DATA-POINTER, by the USAGE IS POINTER clause. If ALL is specified in the VALUE phrase, it is as if all of the categories listed in the syntax for *category-name* were specified.

4. Whether *identifier-1* references an elementary item or a group item, the effect of the execution of the INITIALIZE statement is as though a series of implicit MOVE or SET statements, each of which has an elementary data item as its receiving-operand, were executed. The receiving-operands of these implicit statements are defined in general rule 5 and the sending-operands are defined in general rule 6.

   If the category of a receiving operand is data-pointer, the implicit statement is:

   ```
   SET receiving-operand TO sending-operand
   ```

   Otherwise, the implicit statement is:

   ```
   MOVE sending-operand TO receiving-operand
   ```

5. The receiving operand in each implicit MOVE or SET statement is determined by applying the following steps in order:

   a. First, an elementary data item is a possible receiving item if:

      1) It is explicitly referenced by *identifier-1*; or

      2) It is contained within the group data item referenced by *identifier-1*. If the elementary data item is a table element, each occurrence of the elementary data item is a possible receiving-operand.

   b. Second, the following data items are excluded as receiving-operands:

      1) Any identifiers that are not valid receiving operands of a MOVE statement, except data items of category data-pointer. (For example, index data items are excluded as receiving-operands.)

      2) If the FILLER phrase is not specified, elementary data items with an explicit or implicit FILLER clause. If the FILLER phrase is specified, elementary

data items with an explicit or implicit FILLER clause are not excluded and may be initialized by the INITIALIZE statement.

3) Any elementary data item subordinate to *identifier-1* whose data description entry contains a REDEFINES clause or is subordinate to a data item whose data description entry contains a REDEFINES clause. However, *identifier-1* may itself have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause.

c. Finally, each non-excluded possible receiving-operand is a receiving item if at least one of the following is true:

1) The VALUE phrase is specified, the category of the elementary data item is one of the categories specified or implied in the VALUE phrase, and the VALUE clause is specified in the data description entry of the elementary data item.

2) The REPLACING phrase is specified and the category of the elementary data item is one of the categories specified in the REPLACING phrase.

3) The DEFAULT phrase is specified or neither the REPLACING nor the VALUE phrase is specified.

6. The sending-operand in each implicit MOVE or SET statement is determined as follows:

a. If the data item being initialized qualifies as a receiving-operand because of the VALUE phrase, the sending-operand is determined by the literal in the VALUE clause specified in the data description entry of the receiving-operand data item. If the data item is a table element, the literal in the VALUE clause that corresponds to the occurrence being initialized determines the sending-operand. For categories other than data-pointer, the actual sending-operand is a literal that, when moved to the receiving-operand with a MOVE statement, produces the same result as the initial value of the data item as produced by the application of the VALUE clause.

b. If the data item being initialized does not qualify as a receiving-operand because of the VALUE phrase, but does qualify because of the REPLACING phrase, the sending-operand is the *literal-1* or *identifier-2* associated with the category specified in the REPLACING phrase that matched the category of the receiving-operand.

c. If the data item does not qualify as a receiving-operand because of the VALUE or REPLACING phrases, the sending-operand used depends on the category of the receiving-operand as shown in Table 6-5.

Table 6-5: Default Initialization Values

Category of Receiving-Operand	Sending-Operand
Alphabetic	SPACES
Alphanumeric	SPACES
Alphanumeric-edited	SPACES
Data-pointer	NULL
Numeric	ZERO
Numeric-edited	ZERO

7. The order of execution of these implicit MOVE and SET statements is the order, left to right, of the specification of each *identifier-1* in the INITIALIZE statement. Within this sequence, whenever *identifier-1* refers to a group data item, affected elementary data items are initialized in the sequence of their definition within the group data item. For a fixed-occurrence data item, all occurrences of the affected elementary data items are initialized. For a variable-occurrence data item, the number of occurrences initialized is determined by the rules of the OCCURS clause for a receiving data item.

8. If *identifier-1* occupies the same storage area as *identifier-2*, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. See "Overlapping Operands" on page 5-19 for additional information.

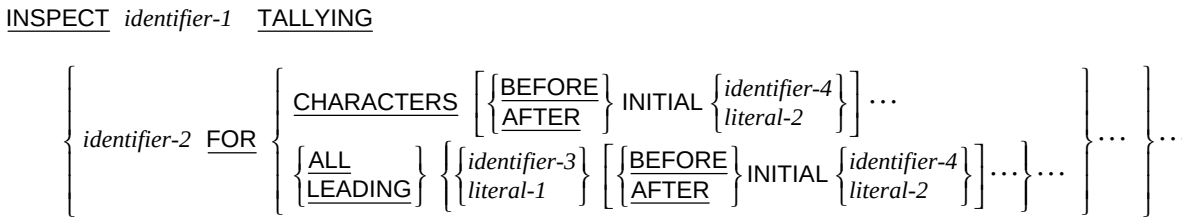
INITIALIZE Statement Examples

```
INITIALIZE EMPLOYEE-RECORD HR-RECORD.


INITIALIZE EMPLOYEE-RECORD
  REPLACING NUMERIC DATA BY ZERO
            ALPHANUMERIC DATA BY ALL "#".


INITIALIZE HR-RECORD
  REPLACING NUMERIC DATA BY 100.00.


INITIALIZE EMPLOYEE-RECORD HR-RECORD
  WITH FILLER
  ALL TO VALUE
  THEN REPLACING
    ALPHANUMERIC ALPHABETIC DATA BY ALL "#"
  THEN TO DEFAULT.
```

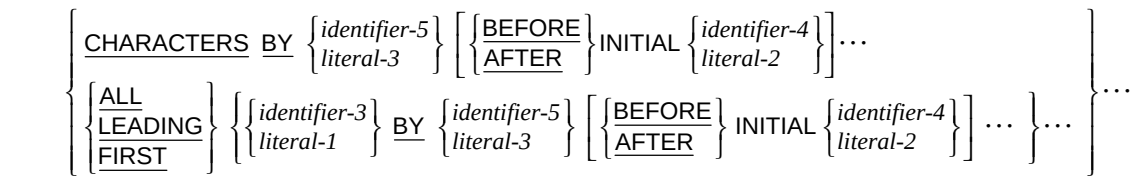
INSPECT Statement

The INSPECT statement provides the ability to tally (Format 1), replace (Format 2), or tally and replace (Format 3) occurrences of single characters or groups of characters in a data item.

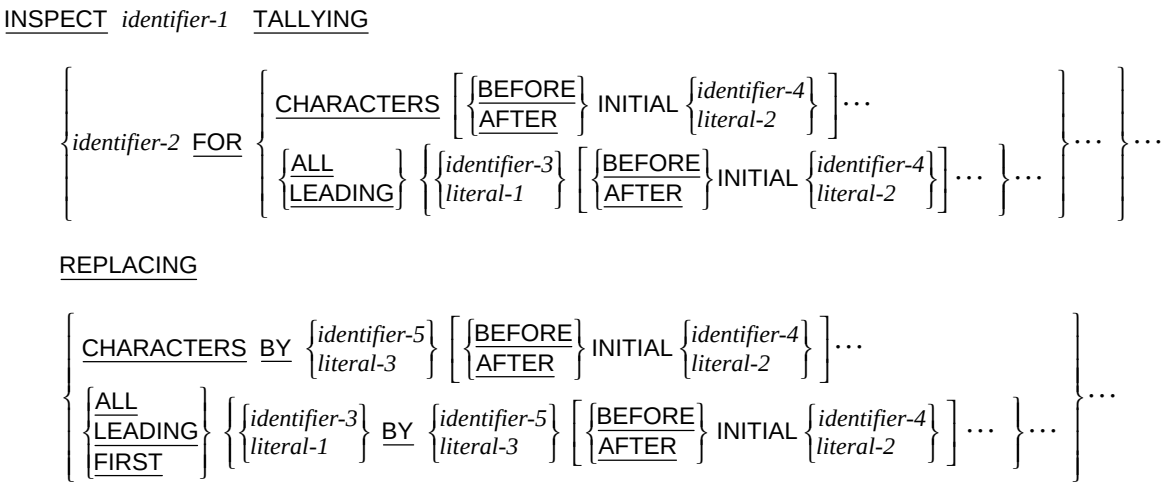
Format 1: Inspect…Tallying



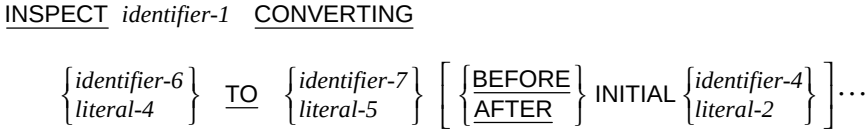
Format 2: Inspect…Replacing



Format 3: Inspect…Tallying…Replacing



Format 4: Inspect…Converting



identifier-1 must reference either a group item or any category of elementary items that have DISPLAY usage.

identifier-3, . . ., *identifier-n* must reference an elementary item that has DISPLAY usage.

Each literal must be a nonnumeric literal and may be any figurative constant except those that begin with the word ALL. If *literal-1*, *literal-2* or *literal-4* is a figurative constant, it refers to an implicit one-character data item.

No more than one BEFORE phrase and one AFTER phrase can be specified for any one ALL, LEADING, CHARACTERS, FIRST or CONVERTING phrase.

For Formats 1 and 3:

- *identifier-2* must reference an elementary numeric data item.

For Formats 2 and 3:

- The size of *literal-3* or the data item referenced by *identifier-5* must be equal to the size of *literal-1* or the data item referenced by *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 to the size of the data item referenced by *identifier-3*.

- 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.

For Format 4:

- 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*, its size is equal to the size of *literal-4* or to the size of the data item referenced by *identifier-6*.
- 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 general rules that apply to the INSPECT statement are as follows:

1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying, replacing, or both) 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 through 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* is treated as follows:
    a. If any *identifier-1*, *identifier-3*, *identifier-4*, *identifier-5*, *identifier-6*, or *identifier-7* refers to an alphabetic or alphanumeric data item, the INSPECT statement treats the contents of each such data item as a character-string.
    b. If any *identifier-1*, *identifier-3*, *identifier-4*, *identifier-5*, *identifier-6* or *identifier-7* refers to an alphanumeric edited, numeric edited or unsigned numeric data item, the data item is inspected as though it had been redefined as alphanumeric (see general rule 2a) and the INSPECT statement had been written to reference the redefined data item.
    c. If any *identifier-1*, *identifier-3*, *identifier-4*, *identifier-5*, *identifier-6* or *identifier-7* refers to a signed numeric data item, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length (excluding any separate sign position) and then the rules in general rule 2b had been applied. (See the discussion of the MOVE statement that begins on page 6-113.) If *identifier-1* is a signed numeric item, the original value of the sign is retained upon completion of the INSPECT statement.

3. In general rules 5 through 17, 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.

4. Subscripting associated with any identifier is evaluated only once as the first operation in the execution of the INSPECT statement.

5. During inspection of the contents of the data item referenced by *identifier-1*, each properly matched occurrence of *literal-1* is tallied (Formats 1 and 3) or replaced by *literal-3* (Formats 2 and 3).

6. 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 phrase are considered in the order they are specified in the INSPECT statement from left to right. The first *literal-1* is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by *identifier-1*. *literal-1* matches that portion of the contents of the data item referenced by *identifier-1* if they are equal, character for character, and if any of the following conditions are present:

      (1) Neither LEADING nor FIRST is specified.

      (2) The LEADING adjective applies to *literal-1* and *literal-1* is a leading occurrence as defined in general rules 10 and 13.

      (3) The FIRST adjective applies to *literal-1* and *literal-1* is the first occurrence as defined in general rule 13.

   b. If no match occurs in the comparison of the first *literal-1*, the comparison is repeated with each successive *literal-1*, if any, until either a match is found or there is no next successive *literal-1*. In the latter case, 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 the new leftmost character position, and the comparison cycle begins again with the first *literal-1*.

   c. Whenever a match occurs, tallying or replacing takes place as described in general rules 10 and 13. The character position in the data item referenced by *identifier-1* immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by *identifier-1*, and the comparison cycle starts again with the first *literal-1*.

   d. The comparison operation continues until the rightmost character position of the data item referenced by *identifier-1* has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

e. If the CHARACTERS phrase is specified, an implied one-character operand participates in the cycle described in general rules 6a through 6d above as if it had been specified as *literal-1*, 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.

7. The comparison operation defined in general rule 6 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 participates in the comparison operation as described in general rule 6. *literal-1* or the implied operand of the CHARACTERS phrase is first eligible to participate in matching at the leftmost character position of the data item referenced by *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 which involve that portion of the contents of the data item referenced by *identifier-1* from its leftmost character position up to, but not including, the first occurrence of *literal-2* within the contents of the data item referenced by *identifier-1*. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 6 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* 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*. This is the character position at which *literal-1* or the implied operand of the CHARACTERS phrase is first eligible to participate in matching. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 6 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.

For Format 1

8. The required words ALL and LEADING are adjectives that apply to each succeeding *literal-1* until the next adjective appears.

9. The contents of the data item referenced by *identifier-2* is not initialized before the execution of the INSPECT statement.

10. 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* 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 the sense of general rule 6e, within the contents of the data item referenced by *identifier-1*.

11. 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 it is defined by the same data description entry.

For Format 2:

12. The required words ALL, LEADING and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.

13. The rules for replacement are as follows:

    a. When the CHARACTERS phrase is specified, each character matched, in the sense of general rule 6e, 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 successive contiguous occurrence of *literal-1* matched in the contents of the data item referenced by *identifier-1* is replaced by *literal-3*, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which *literal-1* was eligible to participate.

d. When the adjective FIRST is specified, the leftmost occurrence of *literal-1* matched within the contents of the data item referenced by *identifier-1* is replaced by *literal-3*. This rule applies to each successive specification of the FIRST phrase regardless of the value of *literal-1*.

14. 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 it is defined by the same data description entry.

For Format 3:

15. A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same *identifier-1* had been written with the first statement being a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement, and the second statement being a Format 2 statement with REPLACING phrases identical to those specified in the Format 3 statement. The general rules given for matching and counting apply to the Format 1 statement and the general rules given for matching and replacing apply to the Format 2 statement. If any of the identifiers in the Format 2 statement are subscripted, their subscripts are evaluated only once before executing the Format 1 statement.

For Format 4:

16. 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.

17. 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 it is defined by the same data description entry.

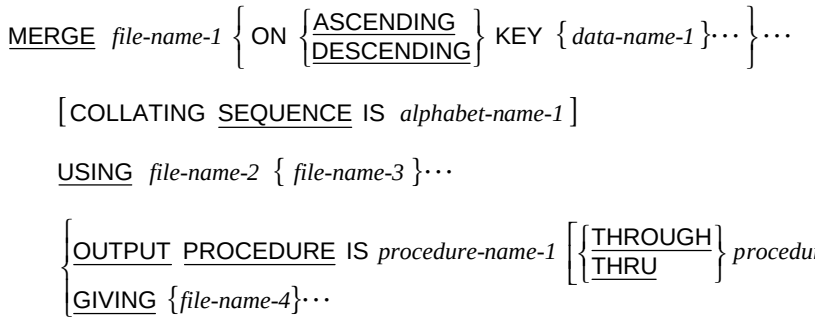
INSPECT Statement Examples

```
      MOVE ZERO TO COUNT-1, COUNT-2.
      INSPECT WORD-1 TALLYING
        COUNT-1 FOR LEADING "L" BEFORE INITIAL "A"
        COUNT-2 FOR LEADING "A" BEFORE INITIAL "L".

*> WORD-1 = "LARGE"   -> COUNT-1 = 1, COUNT-2 = 0
*> WORD-1 = "ANALYST" -> COUNT-1 = 0, COUNT-2 = 1



      MOVE ZERO TO COUNT-1.
      INSPECT WORD-1 TALLYING
        COUNT-1 FOR ALL "L" REPLACING
        ALL "A" BY "E" AFTER INITIAL "L".

*> WORD-1 = "CALLAR" -> COUNT-1 = 2, WORD-1 = "CALLER"
*> WORD-1 = "SALAMI" -> COUNT-1 = 1, WORD-1 = "SALEMI"
*> WORD-1 = "LATTER" -> COUNT-1 = 1, WORD-1 = "LETTER"



      INSPECT WORD-1 REPLACING
        ALL "A" BY "G" BEFORE INITIAL "X".

*> WORD-1 = "ARXAX"  -> WORD-1 = "GRXAX"
*> WORD-1 = "HANDAX" -> WORD-1 = "HGNDGX"



      MOVE ZERO TO COUNT-1.
      INSPECT WORD-1 TALLYING
        COUNT-1 FOR CHARACTERS AFTER INITIAL "J"
        REPLACING ALL "A" BY "B".

*> WORD-1 = "ADJECTIVE" -> COUNT-1 = 6, WORD-1 = "BDJECTIVE"



      INSPECT WORD-1 REPLACING ALL "X" BY "Y",
        "B" BY "Z", "W" BY "Q" AFTER INITIAL "R".

*> WORD-1 = "RXXBQWY"  -> WORD-1 = "RYYZQQY"
*> WORD-1 = "YZACDWBR" -> WORD-1 = "YZACDWZR"
*> WORD-1 = "RAWRXEB"  -> WORD-1 = "RAQRYEZ"
```

```
     INSPECT WORD-1 REPLACING CHARACTERS BY "B"
       BEFORE INITIAL "A".

*> WORD-1 = "12 XZABCD" -> WORD-1 = "BBBBBABCD"
*> WORD-1 = "123456789" -> WORD-1 = "BBBBBBBBB"
*> WORD-1 = "A23456789" -> WORD-1 = "A23456789"



     INSPECT WORD-1 CONVERTING
       "abcdefghijklmnopqrstuvwxyz" TO
       "ABCDEFGHIJKLMNOPQRSTUVWXYZ".

*> WORD-1 = "name"      -> WORD-1 = "NAME"
*> WORD-1 = "Day Total" -> WORD-1 = "DAY TOTAL"
```

MERGE Statement

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merged order, to an output procedure or to an output file.



A MERGE statement may appear anywhere in the Procedure Division except in the declaratives portion.

file-name-1 must be described in a sort-merge file description entry in the Data Division.

data-name-1 may be qualified. *data-name-1* must reference either a record-name associated with *file-name-1* or a data item in a record associated with *file-name-1*. If more than one record description entry is associated with *file-name-1*, the data items referenced by different specifications of *data-name-1* need not all be associated with the same record description entry.

The data item referenced by *data-name-1* must not be a group item that contains a variable-occurrence data item.

file-name-2, *file-name-3*, and *file-name-4* must be described in a file description entry in the Data Division.

No two files specified in any one MERGE statement may reside on the same multiple file reel (or reels). (See the discussion of the I-O-CONTROL paragraph that begins on page 3-36.)

File-names must not be repeated within the MERGE statement.

The words THRU and THROUGH are synonymous.

No pair of file-names in a MERGE statement may be specified in the same SAME AREA, SAME RECORD AREA, SAME SORT AREA or SAME SORT-MERGE AREA clause. (See the I-O-CONTROL paragraph.)

If the file referenced by *file-name-1* contains variable-length records, the size of the records contained in the files referenced by *file-name-2* and *file-name-3* must not be shorter than the shortest record nor longer than the longest 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 files referenced by *file-name-2* and *file-name-3* must not be longer than the longest record described for *file-name-1*.

If the GIVING phrase is specified and the file referenced by *file-name-4* contains variable-length records, the size of the records contained in the file referenced by *file-name-1* must not be shorter than the shortest record nor longer than the longest record size specified for *file-name-4*. If the file referenced by *file-name-4* contains fixed-length records, the size of the records contained in the file referenced by *file-name-1* must not be longer than the fixed record size specified for *file-name-4*.

General Rules

The general rules applying to the MERGE statement are as follows:

1. The MERGE statement merges all records contained in the files referenced by *file-name-2* and *file-name-3* and returns them to an output procedure, or to the file referenced by *file-name-4*, in an order determined by the ASCENDING and DESCENDING phrases and the values of the data items referenced by the specifications of *data-name-1*.

2. The words ASCENDING and DESCENDING apply to each subsequent occurrence of *data-name-1* until another word ASCENDING or DESCENDING is encountered.

3. The data items referenced by the specification of *data-name-1* are the key data items that determine the order in which records are returned from the file referenced by *file-name-1*. The order of significance of the keys is the order in which they are specified in the MERGE statement, without regard to their association with ASCENDING or DESCENDING phrases. The first (or only) key data item is the most significant. Further key data items, if any, are of progressively lesser significance.

4. To determine the relative order in which two records are returned from the file referenced by *file-name-1*, the contents of corresponding key data items are compared according to the rules for comparison of operands in a relation condition, starting with the most significant key data item.

    a. If the contents of the corresponding key data items are not equal and the key is associated with the ASCENDING phrase, the record containing the key data item with the lower value is returned first.

    b. If the contents of the corresponding key data item are not equal and the key is associated with the DESCENDING phrase, the record containing the key data item with the higher value is returned first.

    c. If the contents of the corresponding key data items are equal, the determination is made on the contents of the next most significant key data item.

    d. If the contents of all the key data items in one record are equal to the contents of the corresponding key data items in another record, the records are returned in the order in which their associated input files are specified in the MERGE statement. If both records are associated with the same file, the order of the records in that file is preserved.

5. The collating sequence that applies to the comparison of nonnumeric key data items is determined at the beginning of the execution of the MERGE statement in the following order of precedence:

    a. The collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement

    b. The collating sequence established as the program collating sequence

6. The results of the merge operation are undefined unless the records in the files referenced by *file-name-2* and *file-name-3* are ordered as described in the ASCENDING or DESCENDING KEY clauses associated with the MERGE statement.

7. All the records in the files referenced by *file-name-2* and *file-name-3* in the USING phrase are transferred to the file referenced by *file-name-1*. At the start of execution of the MERGE statement, the files referenced by *file-name-2* and *file-name-3* must not be in the open mode. For each of the files referenced by *file-name-2* and *file-name-3*, the execution of the MERGE statement causes the following actions to be taken:

    a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase had been executed. If an output procedure is specified, this initiation is performed before control passes to the output procedure.

b. The logical records are obtained and released to the merge operation. Each record is obtained as if a READ statement with the NEXT and the AT END phrases had been executed. If the file referenced by *file-name-1* contains fixed-length records, any record in the files referenced by *file-name-2* and *file-name-3* containing fewer character positions than that specified for *file-name-1* is space-filled on the right beginning with the first character position after the last character in the record when that record is released to the file referenced by *file-name-1*.

c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. If an output procedure is specified, this termination is not performed until after control passes the last statement in the output procedure.

These implicit functions are performed such that any associated USE AFTER EXCEPTION procedures are executed.

8. During the execution of any USE AFTER EXCEPTION procedure implicitly invoked while executing the MERGE statement, no statements may be executed which manipulate the file referenced by *file-name-2*, *file-name-3*, or *file-name-4*, or which access the record area associated with *file-name-2*, *file-name-3*, or *file-name-4*.

9. The output procedure may consist of any procedure needed to select, modify or copy the records that are made available one at a time by the RETURN statement in merged order from the file referenced by *file-name-1*. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT without the optional PROGRAM phrase, GO TO and PERFORM statements in the range of the output procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure. The range of the output procedure must not cause the execution of any MERGE, RELEASE or SORT statement.

10. If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last statement 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 statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it can select the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

11. During the execution of the output procedure, no statement may be executed manipulating the file referenced by, or accessing the record area associated with, *file-name-2* or *file-name-3*.

12. If the GIVING phrase is specified, all the merged records are written on the file referenced by *file-name-4* 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-4* must not be in the open mode. For each of the files referenced by *file-name-4*, the execution of the MERGE statement causes the following actions to be taken:

    a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT 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.

    c. 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; and so forth. After execution of the MERGE statement, the contents of the relative key data item indicate the last record returned to the file.

    d. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

    These implicit functions are performed such that any associated USE procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, *file-name-4*. On the first attempt to write beyond the externally defined boundaries of the file, any USE procedure specified for the file is executed; if control is returned from that USE procedure or if no such USE procedure is specified, the processing of the file is terminated as described above.

13. 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, any output procedure referenced by that MERGE statement must appear:

        (1) Totally within nonindependent segments, or

        (2) Wholly contained in a single independent segment.

    b. If a MERGE statement appears in an independent segment, any output procedure referenced by that MERGE statement must be contained:

        (1) Totally within nonindependent segments, or

        (2) Wholly within the same independent segment as that MERGE statement.

MERGE Statement Example

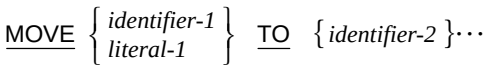
```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  MERGE01.
*
*  Examples for RM/COBOL Language Reference Manaual.
*  MERGE statement.
*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT MERGE-FILE ASSIGN TO SORT-WORK.
     SELECT SORTED-FILE-1 ASSIGN TO DISK.
     SELECT SORTED-FILE-2 ASSIGN TO DISK.



 DATA DIVISION.
 FILE SECTION.
 SD MERGE-FILE.
 01 MERGE-RECORD.
    02 MERGE-KEY-1        PIC X(05).
    02 MERGE-KEY-2        PIC 9(05) BINARY.
    02 MERGE-DATA-1       PIC X(20).
 FD SORTED-FILE-1.
 01 SORTED-FILE-1-RECORD.
    02 SORTED-KEY-1       PIC X(05).
    02 SORTED-KEY-2       PIC 9(05) BINARY.
    02 SORTED-DATA-1      PIC X(20).
 FD SORTED-FILE-2.
 01 SORTED-FILE-2-RECORD.
    02 SORTED-KEY-1       PIC X(05).
    02 SORTED-KEY-2       PIC 9(05) BINARY.
    02 SORTED-DATA-1      PIC X(20).
 WORKING-STORAGE SECTION.
 01 EOF-FLAG              PIC X(01).
    88 EOF                VALUE "T" WHEN FALSE "F".
```

```
PROCEDURE DIVISION.
MAIN1.
    MERGE MERGE-FILE
      ON ASCENDING KEY MERGE-KEY-1
      ON DESCENDING KEY MERGE-KEY-2
      USING SORTED-FILE-1 SORTED-FILE-2
      OUTPUT PROCEDURE IS PUT-RECORDS.
    STOP RUN.

PUT-RECORDS.
    SET EOF TO FALSE.
    PERFORM UNTIL EOF
      RETURN MERGE-FILE RECORD
      AT END SET EOF TO TRUE
      NOT AT END CALL "WRITE-RECORD" USING MERGE-RECORD
      END-RETURN
    END-PERFORM.

END PROGRAM MERGE01.
```

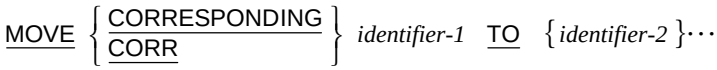
MOVE Statement

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

Format 1: Move…To



Format 2: Move Corresponding



literal-1 or the data item referenced by *identifier-1* represents the sending area; *identifier-2* (. . .) represents the receiving area (or areas).

An index data item must not appear as an operand of a MOVE statement.

The data designated by *literal-1* or *identifier-1* is moved to the data item referenced by each *identifier-2* in the order in which it is specified. The rules referring to *identifier-2* also apply to the other receiving areas.

Any length evaluation or subscripting associated with *identifier-2* is evaluated immediately before the data is moved to the respective data item. Any length evaluation or subscripting associated with *identifier-1* is evaluated only once, 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).
```

Any move in which the receiving operand is an elementary item and the sending operand is either a literal or an elementary item is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited or alphanumeric edited. These categories are described in the PICTURE clause. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO, when moved to a numeric or numeric edited item, belongs to the category numeric; in all other cases it belongs to the

category alphanumeric. 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 between these categories:

1. The figurative constant SPACE, or an alphanumeric edited or alphabetic data item must not be moved to a numeric or numeric edited data item.

2. 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.

3. A noninteger numeric literal or a noninteger numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

4. All other elementary moves are legal and are performed according to the rules given below.

Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any de-editing implied by the sending data item or editing specified for the receiving data item:

1. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space-filling takes place as defined in the discussion of standard alignment rules that begins on page 4-96. 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 is not moved; if the operational sign occupies a separate character position (see the discussion of the SIGN clause on page 4-94), that character is not moved and the size of the sending item is considered to be one less than its actual size (in terms of standard data format characters). If the sending item is numeric edited, no de-editing takes place. If the usage of the sending operand is different from that of the receiving operand, conversion of the sending operand to the internal representation of the receiving operand takes place. If the PICTURE character-string of the sending operand contains the symbol “P”, all digit positions specified with this symbol are considered to contain the value zero and are counted in the size of the sending item.

2. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero filling takes place (see the discussion of standard alignment rules) where zeroes are replaced because of editing requirements.

   When a signed item is the receiving item, the sign of the sending item is placed in the receiving item (see the discussion of the SIGN clause). 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 the sending operand is described as being alphanumeric, data is moved as if the sending operand were described as an unsigned numeric integer.

When a numeric edited data item is the sending item, conversion is implied to establish the unedited numeric value of the operand, which may be signed; then the unedited numeric value is moved to the receiving field. The implied conversion deletes all characters other than the decimal digits 0, 1, . . . 9, sets the operational sign negative if a minus sign is present in the sending item or positive otherwise, and sets the scale according to the rightmost decimal point present in the sending item or to the scale of the sending data item otherwise. The representation of the decimal point used in this conversion is a period unless the DECIMAL POINT IS COMMA clause is specified in the source program, in which case a comma is used. In this conversion, any decimal digit 0 that matches an inserted character 0 in the sending item is excluded from the resulting unedited numeric value.

3. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place (see the discussion of standard alignment rules that begins on page 4-96).

   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.

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 is filled without regard for the individual elementary or group items contained within either the sending or receiving area, except as noted in the OCCURS clause. When a group item is moved to an elementary item described with the JUSTIFIED RIGHT clause, right justification occurs.

When a sending and receiving item share a part of their storage areas, the result of the execution of such a statement is undefined.

Table 6-6 summarizes the legality of the various types of MOVE statements.

Table 6-6: Types of MOVE Statements and Their Legality

Sending Data Item	Category of Receiving Data Items		
	Alphabetic	Alphanumeric Edited Alphanumeric	Numeric Integer Numeric Noninteger Numeric Edited
Alphabetic	■	■	□
Alphanumeric	■	■	■
Alphanumeric Edited	■	■	□
Numeric Integer	□	■	■
Numeric Noninteger	□	□	■
Numeric Edited	□	■	■

■ *Allowed.*

□ *Disallowed.*

CORRESPONDING Phrase

$$\left\{\begin{array}{l}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{array}\right\}\ \textit{identifier-1}\ \ \underline{\text{TO}}\ \ \{\textit{identifier-2}\}\cdots$$

When the CORRESPONDING phrase is specified, all identifiers must refer to group items. When a MOVE statement with a CORRESPONDING phrase specifies more than one receiving group item (*identifier-2*), the effect is the same as if multiple MOVE statements with CORRESPONDING phrases had been written, one for each of the receiving group items (*identifier-2*), and each having the same sending group item (*identifier-1*).

For the MOVE statement with the CORRESPONDING phrase:

- The description of *identifier-1* and *identifier-2* must not contain level-number 66, 77, 78, or 88 or the USAGE IS INDEX clause.
- Neither *identifier-1* nor *identifier-2* may be reference modified.

- *identifier-1* or *identifier-2* may be described with the OCCURS or REDEFINES clauses or may be subordinate to data items described with the OCCURS or REDEFINES clauses. If *identifier-1* or *identifier-2* is a table element, then the required subscripting must be specified as part of *identifier-1* or *identifier-2*. The specified subscripting will be applied to the selected subordinate corresponding data items, respectively, for *identifier-1* and *identifier-2*.

For each individual MOVE statement with a CORRESPONDING phrase, subordinate data item pairs are selected, one from the sending group item and one from the receiving group item. Then for each such selected pair, data movement occurs from the data item that is subordinate to the sending group item to the data item that is subordinate to the receiving group item. The data movement that occurs is the same as if individual MOVE statements had been written for each of the selected pairs.

The rules that govern the selection of eligible subordinate data item pairs are as follows:

1. The data items are not designated by the keyword FILLER and have the same *data-name* and the same qualifiers up to but not including the original group items, *identifier-1* and *identifier-2*.

2. At least one of the data items is an elementary data item and the resulting move is legal according to the move rules.

3. A data item that is subordinate to *identifier-1* or *identifier-2* 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.

4. The name of each data item that satisfies the above conditions must be unique after application of the implied qualifiers.

When multiple receiving group identifiers (*identifier-2*, . . .) are listed, all corresponding items in the first *identifier-2* are moved prior to moving corresponding items in the second and any subsequent receiving group identifiers.

CORRESPONDING and CORR are synonymous.

MOVE Statement Examples

```
MOVE INCOME TO TOTAL-INCOME.


MOVE 1 TO PAGE-COUNT, LINE-NUM.


MOVE "Marmack Industries" to TITLE-HEADER.


MOVE PERSON IN FILE-RECORD TO
  PERSON OF ALABAMA (I-A OF ALABAMA),
  PERSON OF CROSS-CENSUS.


MOVE NUM TO NUM-ED.


MOVE TABLE-ELT (N, 1, M) TO NEXT-ENTRY
  PREVIOUS-ENTRY.


MOVE -36.7 TO DEFICIT.


MOVE QUOTES TO SECTION-DIVIDER.


MOVE ZERO TO COUN-TER.


MOVE ZEROES TO COUN-TER, NUM, NUM-ED.
```

MULTIPLY Statement

The MULTIPLY statement causes numeric data items to be multiplied and stores the result.

Format 1: Multiply…By

```
MULTIPLY { identifier-1 } BY { identifier-2 [ ROUNDED ] }...
         { literal-1    }

    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ]

    [ END-MULTIPLY ]
```

Format 2: Multiply…Giving

```
MULTIPLY { identifier-1 } BY { identifier-2 }
         { literal-1    }    { literal-2    }

    GIVING { identifier-3 [ ROUNDED ] }...

    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ]

    [ END-MULTIPLY ]
```

In Format 1, the value of *identifier-1* or *literal-1* is multiplied by the value of each *identifier-2*. The value of each multiplier *(identifier-2)* is replaced by this product.

In Format 2, the value of *identifier-1* or *literal-1* is multiplied by *identifier-2* or *literal-2* and the result is stored in each *identifier-3.*

Each identifier must refer to a numeric elementary item, except that in Format 2 the identifiers following the word GIVING may refer to either an elementary numeric item or an elementary numeric edited item.

Each literal must be a numeric literal.

Additional rules and explanations regarding features of the MULTIPLY statement that are common to other arithmetic statements can be found in the discussion of common rules that begins on page 5-16. See in particular the discussions of the ROUNDED phrase, the

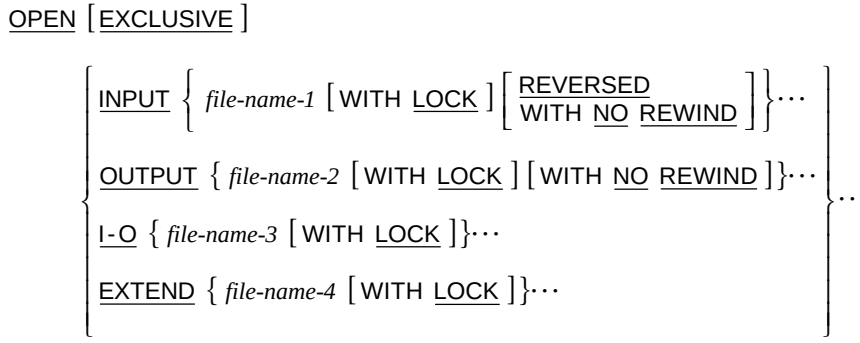
size error condition, overlapping operands, modes of operation, composite size, and incompatible data.

MULTIPLY Statement Examples

```
MULTIPLY 10 BY INCOME.  *> INCOME := (10 * INCOME)


MULTIPLY PRINCIPAL BY INTEREST-RATE
  GIVING INTEREST ROUNDED.


MULTIPLY INFLATION-RATE BY EXPENSES
ON SIZE ERROR
  MOVE 0 TO ECONOMY-RATING
END-MULTIPLY.
```

OPEN Statement

The OPEN statement initiates the processing of files.



The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode. A file is available if it is physically present and recognized by the runtime system. Table 6-7 shows the results of opening available and unavailable files.

The successful execution of an OPEN statement makes the associated record area available to the program. If the file connector associated with the file-name is an external file connector, there is only one record area associated with the file connector for the run unit.

The files referenced in the OPEN statement need not all have the same organization or access.

The EXTEND phrase may only be specified for files with sequential access.

The REVERSED and NO REWIND phrases may only be specified for files that are sequential organization.

The EXCLUSIVE phrase indicates that the open is to obtain exclusive access to each file referenced in the OPEN statement until the file is closed. The EXCLUSIVE phrase is redundant for any file for which the LOCK MODE IS EXCLUSIVE clause is specified in its file control entry.

The LOCK phrase indicates that the open is to obtain exclusive access to the associated file until the file is closed. The LOCK phrase is redundant if the EXCLUSIVE phrase is specified in the same OPEN statement or if the LOCK MODE IS EXCLUSIVE clause is specified in the file control entry for the file.

The successful execution of the OPEN statement sets the lock mode of the file using the EXCLUSIVE and LOCK phrases of the OPEN statement, the LOCK MODE clause, if specified, in the file control entry for the file, or configurable defaults for each open mode. The section "File Locking" on page 5-68 provides a general discussion of lock mode. If the file is opened in shared input-output mode, record locking will apply as described in the section "Record Locking" on page 5-69. The *RM/COBOL User's Guide* contains additional information regarding system dependent features of file and record locking, as well as information on configuration of defaults.

Table 6-7: Availability of a File

File Is	Available	Unavailable
INPUT	Normal open.	Open is unsuccessful.
INPUT (OPTIONAL)	Normal open.	Normal open; first read causes at end condition or invalid key condition.
I–O	Normal open.	Open is unsuccessful.
I–O (OPTIONAL)	Normal open.	Open causes file to be created.
OUTPUT	Normal open; file contains no records.	Open causes file to be created.
EXTEND	Normal open.	Open is unsuccessful.
EXTEND (OPTIONAL)	Normal open.	Open causes file to be created.

Prior to the successful execution of an OPEN statement for a given file, no statement can be executed that references that file, either explicitly or implicitly, except that the file may be listed in the USING or GIVING phrases of a SORT or MERGE statement.

An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In Table 6-8, a ■■■ symbol at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the open mode given at the top of the column.

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 that same file must be preceded by the execution of a CLOSE statement, without the LOCK, REEL or UNIT phrase, for that file.

Table 6-8: Permissible Statements

Access	Statement	Open Mode: Input	Output	I-O	Extend
Sequential	READ	■		■	
	WRITE		■		■
	REWRITE			■	
	START	■		■	
	DELETE			■	
Random	READ	■		■	
	WRITE		■	■	
	REWRITE			■	
	START				
	DELETE			■	
Dynamic	READ	■		■	
	WRITE		■	■	
	REWRITE			■	
	START	■		■	
	DELETE			■	

■ *May be used.*

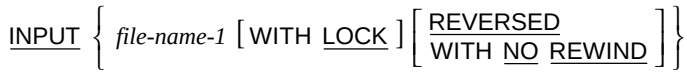
□ *May not be used.*

Execution of the OPEN statement does not obtain or release the first data record.

The file description entry for *file-name-1*, *file-name-3* or *file-name-4* must be equivalent to that used when this file was created.

The execution of an OPEN statement causes the value of the specified file status data item, if any, associated with the file to be updated.

INPUT Phrase



If a file opened with the INPUT phrase is an optional file that is not present, the OPEN statement sets the file position indicator to indicate that an optional input file is not present. Otherwise:

- When sequential or relative files are opened with the INPUT phrase, the file position indicator is set to 1.
- When indexed files are opened with the INPUT phrase, the file position indicator is set to the characters that have the lowest ordinal position in the collating sequence associated with the file, and the prime record key is established as the key of reference.

The REVERSED and NO REWIND phrases may only be specified if *file-name-1* refers to a sequential organization file. Since the NO REWIND phrase is common to the INPUT and OUTPUT phrases, it is discussed separately on page 6-126.

When the REVERSED phrase is specified, the file is positioned at its end by execution of the OPEN statement. Subsequent READ statements for the file make the data records of the file available in reverse order; that is, starting with the last record.

The REVERSED phrase is applicable only to files whose storage medium is capable of reverse motion. It is ignored when not applicable to the storage medium of the file. The *RM/COBOL User's Guide* contains specific information regarding support for the REVERSED phrase.

OUTPUT Phrase

OUTPUT { *file-name-2* [WITH LOCK] [WITH NO REWIND] }

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.

The NO REWIND phrase may only be specified if *file-name-2* refers to a sequential organization file. Since the NO REWIND phrase is common to the INPUT and OUTPUT phrases, it is discussed separately on page 6-126.

I-O Phrase

I-O { *file-name-3* [WITH LOCK] }

The I–O phrase permits the opening of a mass storage file for both input and output operations. If the referenced file does not exist and the OPTIONAL phrase is specified in the SELECT clause for the referenced file, the file is created as a new empty file as is done when the OUTPUT phrase is used.

The I–O phrase can be used only for mass storage files (files assigned to the DISC, DISK, or RANDOM device-type).

When the I–O phrase is specified and the LABEL RECORDS clause indicates that label records are present, the execution of the OPEN statement includes the following:

- The labels are checked.
- New labels are written.

When sequential or relative files are opened with the I–O phrase, the file position indicator is set to 1.

When indexed files are opened with the I–O phrase, the file position indicator is set to the characters that have the lowest ordinal position in the collating sequence associated with the file, and the prime record key is established as the key of reference.

In 1985 mode, if the run unit does not have write access to the file, the execution of an OPEN statement with the I–O phrase is unsuccessful and the I–O status value is set to indicate this condition. In 1974 mode, if the run unit does not have write access to the file, an OPEN statement with the I–O phrase is successful; however, any attempt to execute a DELETE, REWRITE, or WRITE statement while in this mode will be unsuccessful.

EXTEND Phrase

EXTEND { *file-name-4* [WITH LOCK] }

When the EXTEND phrase is specified, the OPEN statement positions the file immediately following the last logical record of that file. Subsequent WRITE statements referencing the file will add records to the file as though the file has been opened with the OUTPUT phrase.

The EXTEND phrase may be specified only if *file-name-4* refers to a file with sequential access. The EXTEND phrase must not be specified for a file whose device-type is INPUT.

The last record for a sequential file is the last record written in the file.

The last record for a relative file is the currently existing record with the highest relative record number.

The last record for an indexed file is the currently existing record with the highest prime key value according to the collating sequence of the file. If the indexed file is described with the DUPLICATES phrase in the RECORD KEY clause of its file control entry and the highest prime key value is duplicated within the records of the file, then the last record is the currently existing record with the highest prime key value that was chronologically last released to the file.

NO REWIND Phrase

WITH NO REWIND

The NO REWIND phrase can be used only with sequential single reel or unit files. The phrase is ignored if it does not apply to the storage medium on which the file resides.

If the storage medium for the file permits rewinding, the following rules apply:

1. When the REVERSED, EXTEND or NO REWIND phrase is not specified, execution of the OPEN statement causes the file to be positioned at its beginning.

2. When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned; the file must be already positioned at its beginning prior to the execution of the OPEN statement.

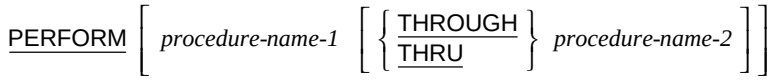
OPEN Statement Examples

```
OPEN EXCLUSIVE INPUT TRANSACTION-FILE.

OPEN EXCLUSIVE OUTPUT LOG-FILE WITH NO REWIND.

OPEN I-O LOG-FILE.

OPEN EXTEND INPUT-FILE.

OPEN INPUT TAPE-FILE-1 REVERSED.

OPEN I-O DATA-BASE WITH LOCK.

OPEN INPUT DATA-BASE.
```

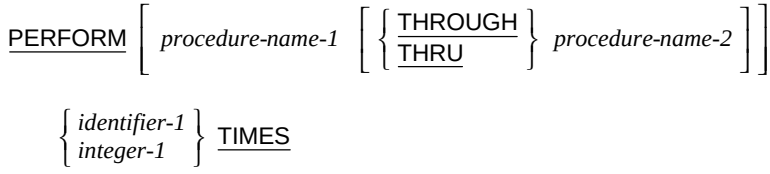
PERFORM Statement

The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete. The PERFORM statement is also used to control execution of one or more imperative statements that are within the scope of that PERFORM statement.

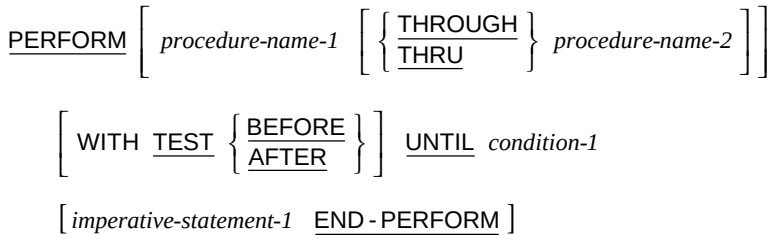
Format 1: Perform (Once)



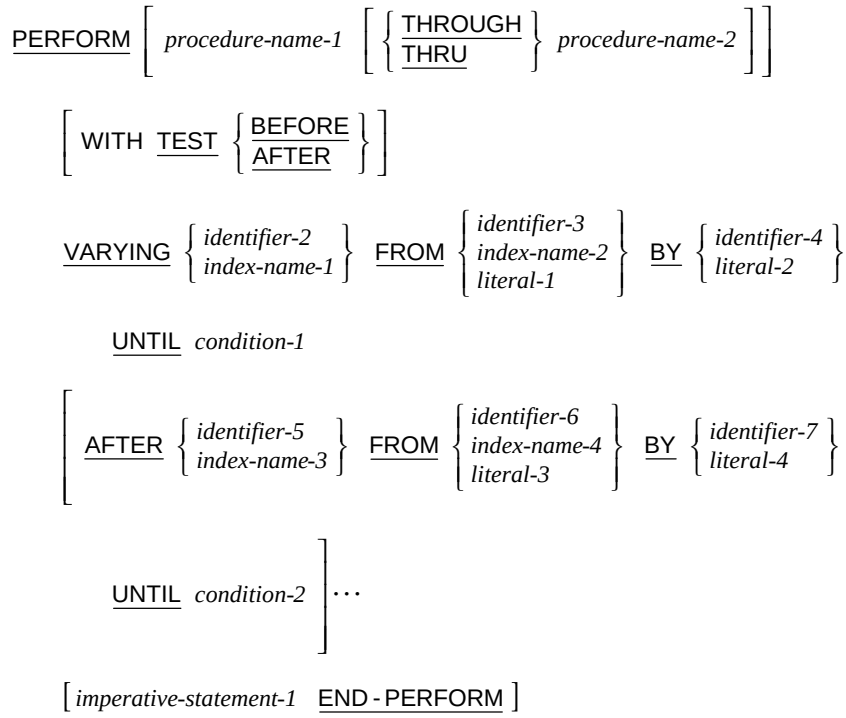
Format 2: Perform…Times



Format 3: Perform…Until



Format 4: Perform…Varying



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.

If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.

Each identifier represents a numeric elementary item described in the Data Division. In Format 2, *identifier-1* must be described as a numeric integer.

Each literal represents a numeric literal.

The words THRU and THROUGH are synonymous.

If an index-name is specified in the VARYING or AFTER phrase, then:

- The identifier in the associated FROM and BY phrases must reference an integer data item.
- The literal in the associated FROM phrase must be a positive integer.
- The literal in the associated BY phrase must be a nonzero integer.

If an index-name is specified in the FROM phrase, then:

- The identifier in the associated VARYING or AFTER phrase must refer to an integer data item.
- The identifier in the associated BY phrase must refer to an integer data item.
- The literal in the associated BY phrase must be an integer.

literal-2 or *literal-4* in the BY phrase must not be zero.

condition-1, *condition-2*, . . . may be any conditional expression.

When *procedure-name-1* and *procedure-name-2* are both specified and either is the name of a procedure in the declaratives portion of the Procedure Division, both must be procedure-names in the same declarative section.

The data items referenced by *identifier-4* and *identifier-7* must not have a zero value.

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.

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.

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.

The END-PERFORM phrase delimits the scope of the in-line PERFORM statement.

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 terms in-line or out-of-line, all the rules

that apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.

When the PERFORM statement is executed, control is transferred to the first statement of the specified set of statements, except as indicated in the general rules for Formats 2 through 4 as given below. This transfer of control occurs only once for each execution of a PERFORM statement. For those cases when 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:

- 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*.
- 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*.
- If *procedure-name-2* is specified and it is a paragraph-name, the return is after the last statement of the paragraph.
- 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.
- If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained within it has been executed.

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 TO 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, *procedure-name-2* may be the name of a paragraph consisting of the EXIT or CONTINUE statement, to which all of these paths must lead.

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.

The PERFORM statements operate as described in the following paragraphs.

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.

Format 2 is the PERFORM . . . TIMES statement. 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 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, references 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*.

Format 3 is the PERFORM . . . UNTIL statement. The specified set of statements is performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the end of the PERFORM statement. If the condition is true when the PERFORM statement is entered and the TEST AFTER phrase is not specified, control passes to the end of the PERFORM statement and the specified set of statements is not executed. In the absence of the TEST AFTER phrase (that is, TEST BEFORE is specified or implied), testing of the specified condition occurs before each execution of the specified set of statements. When the TEST AFTER phrase is specified, the specified set of statements is executed before the specified condition is tested. Any subscripting or reference modification associated with the operands in *condition-1* is evaluated each time the condition is tested.

Format 4 is the PERFORM . . . VARYING statement. This variation of the PERFORM statement is used to augment the values referred to 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 referred to 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*, except that 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 contents of the data item referred to by the identifier are set or augmented. If *identifier-3*, *identifier-4*, *identifier-6* or *identifier-7* is subscripted, the subscripts are evaluated each time the contents of the data item referred to by the identifier are used in a setting or augmenting operation. Any subscripting or reference modification associated with the

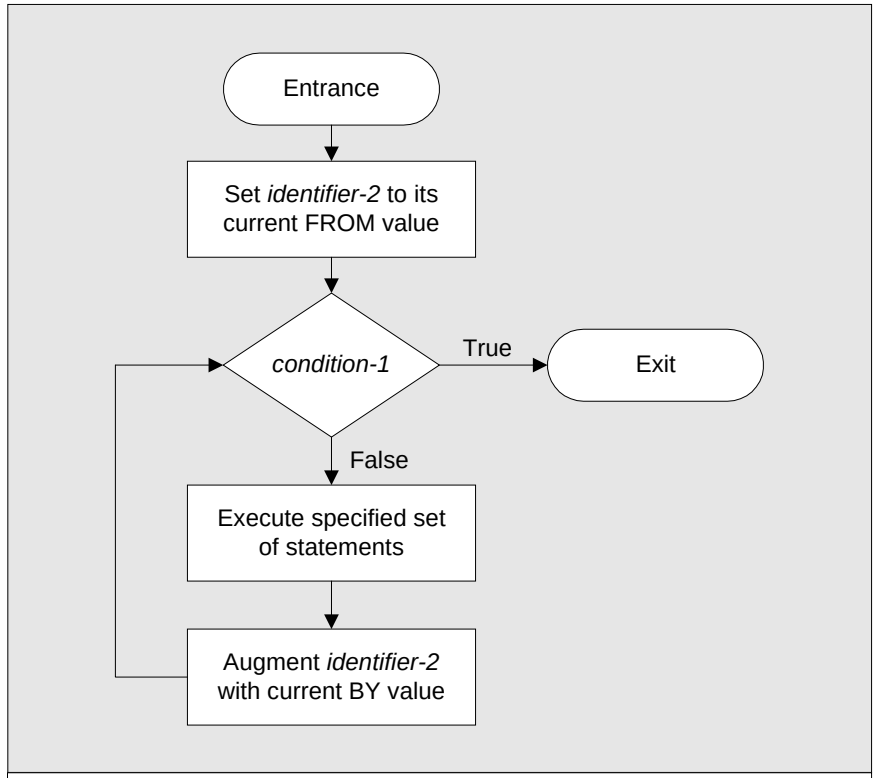
operands specified in *condition-1* or *condition-2* is evaluated each time the condition is tested.

The following paragraphs specify in detail the actions that occur as a result of executing four of the simpler forms of a PERFORM . . . VARYING statement. The actions that occur when more complex forms of the PERFORM . . . VARYING statement are executed may be inferred by generalization from the simpler forms.

For a PERFORM . . . VARYING statement that does not have an AFTER phrase nor a TEST AFTER phrase (that is, TEST BEFORE is specified or implied), the data item referred to by *identifier-2* is set to *literal-1* or the current value of the data item referred to by *identifier-3* at the point of initial execution of the statement; then *condition-1* in the UNTIL phrase is tested. If it is false, the specified set of statements is executed once. The value of the data item referred to by *identifier-2* is augmented by the specified increment or decrement value (*literal-2* or the value of the data item referred to by *identifier-4* in the BY phrase) and *condition-1* is retested, with subsequent execution of the specified set of statements if it is found to be false. The cycle continues until *condition-1* is found to be true, at which time 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 without executing the specified set of statements at all.

Figure 6-1 represents this sequence of actions.

Figure 6-1: PERFORM . . . VARYING Statement



This PERFORM . . . VARYING statement has no AFTER or TEST AFTER phrase.

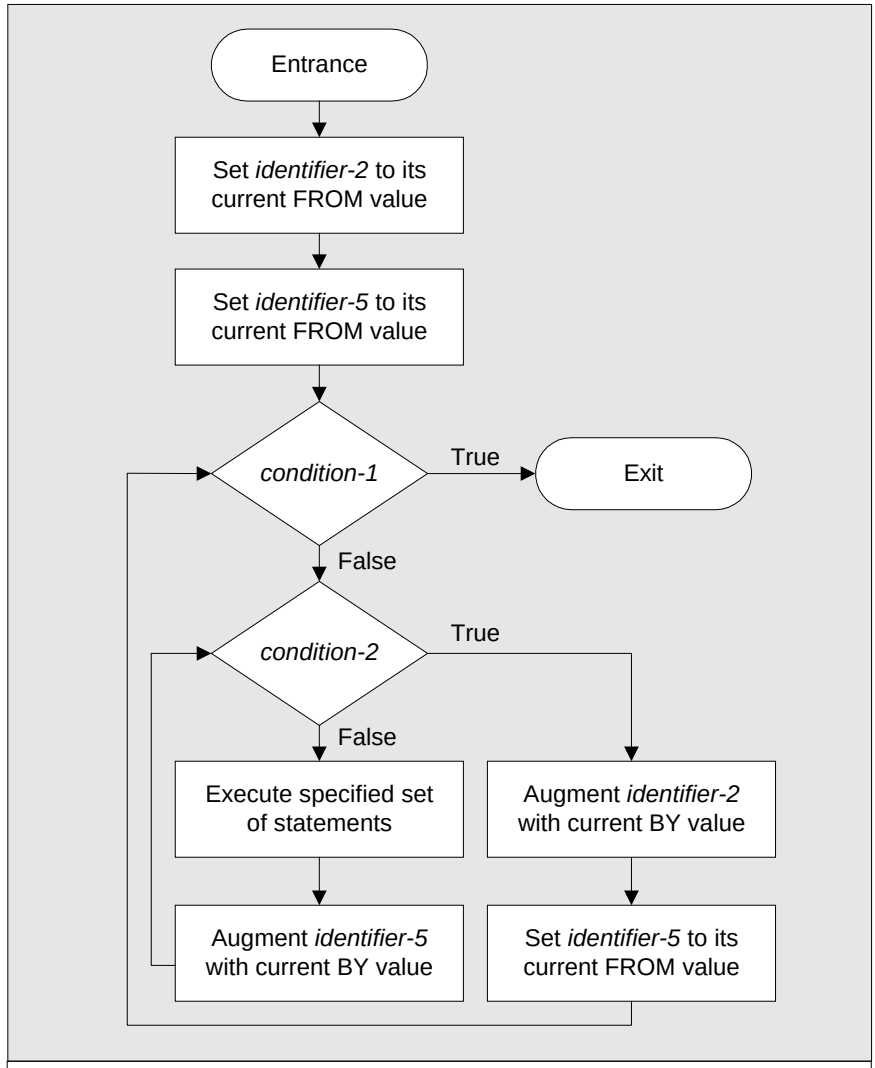
When control reaches the end of this form of the PERFORM . . . VARYING statement, the data item referred to by *identifier-2* contains a value that exceeds the setting last used by one increment or decrement value, unless *condition-1* was true to begin with, in which case it contains *literal-1* or the current value of the data item referred to by *identifier-3*.

For a PERFORM . . . VARYING statement that has one AFTER phrase but no TEST AFTER phrase (that is, TEST BEFORE is specified or implied), the data item referred to by *identifier-2* is set to *literal-1* or to the current value of the data item referred to by *identifier-3*; then the data item referred to by *identifier-5* is set to *literal-3* or to the current value of the data item referred to by *identifier-6*. Subsequent actions form a nested set of two cycles. *condition-1* is tested. If it is true, control is transferred to the end of the PERFORM statement; if it is false, *condition-2* is tested. If *condition-2* is false, the specified set of statements is executed once, then the data item referred to by *identifier-5* is augmented by *literal-4* or by the current value of the data item referred to by *identifier-7*, and *condition-2* is retested with subsequent execution of the specified set

of statements if it is found to be false. This inner cycle of execution, testing, and augmentation continues until *condition-2* is found to be true, at which time the data item referred to by *identifier-2* is augmented by *literal-2* or by the current value of the data item referred to by *identifier-4*, *identifier-5* is set to *literal-3* or to the current value of the data item referred to by *identifier-6*, and *condition-1* is retested with subsequent reevaluation of *condition-2* as long as *condition-1* is found to be false. This outer cycle continues until *condition-1* is found to be true.

Figure 6-2 represents this sequence of actions.

Figure 6-2: PERFORM . . . VARYING Statement



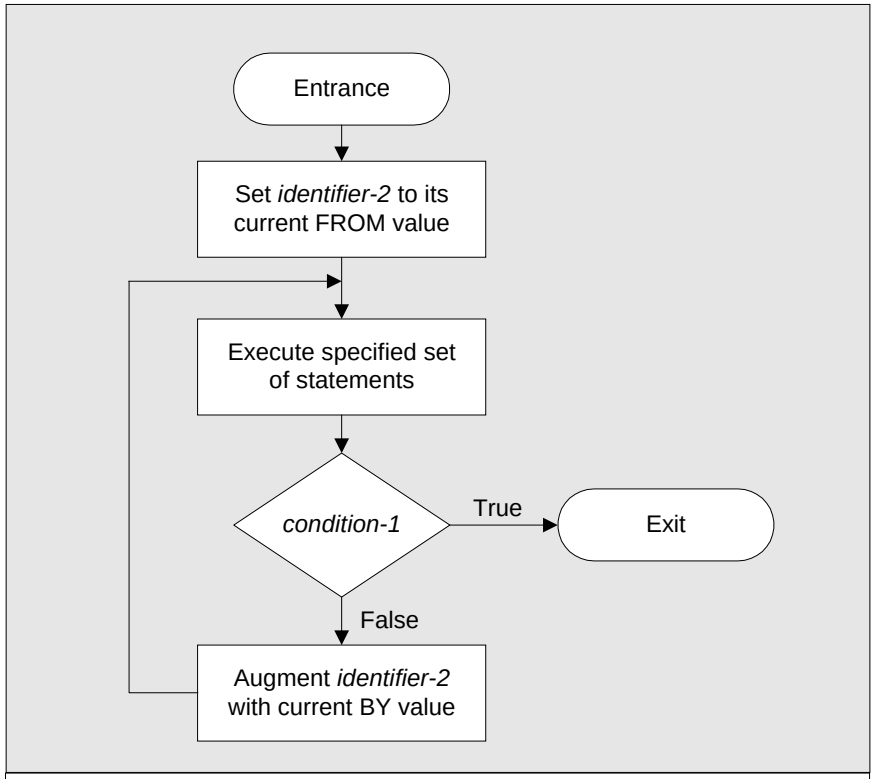
This PERFORM . . . VARYING statement has one AFTER phrase and no TEST AFTER phrase.

When control reaches the end of this form of the PERFORM . . . VARYING statement, the data item referred to by *identifier-5* contains *literal-3* or the current value of the data item referred to by *identifier-6*. The data item referred to by *identifier-2* contains a value that exceeds the last used setting by one increment or decrement value, unless *condition-1* was true to begin with, in which case it contains *literal-1* or the current value of the data item referred to by *identifier-3*.

For a PERFORM . . . VARYING statement that does not have an AFTER phrase but does have a TEST AFTER phrase, the data item referred to by *identifier-2* is set to *literal-1* or the current value of the data item referred to by *identifier-3* at the point of initial execution of the statement; then the specified set of statements is executed once and *condition-1* in the UNTIL phrase is tested. If it is false, the value of the data item referred to by *identifier-2* is augmented by the specified increment or decrement value (*literal-2* or the value of the data item referred to by *identifier-4*) and the specified set of statements is reexecuted with subsequent reevaluation of *condition-1*. The cycle continues until *condition-1* is found to be true, at which time control is transferred to the end of the PERFORM statement. For this form of the PERFORM . . . VARYING statement, the specified set of statements is always executed at least once.

Figure 6-3 represents this sequence of actions.

Figure 6-3: PERFORM . . . VARYING Statement



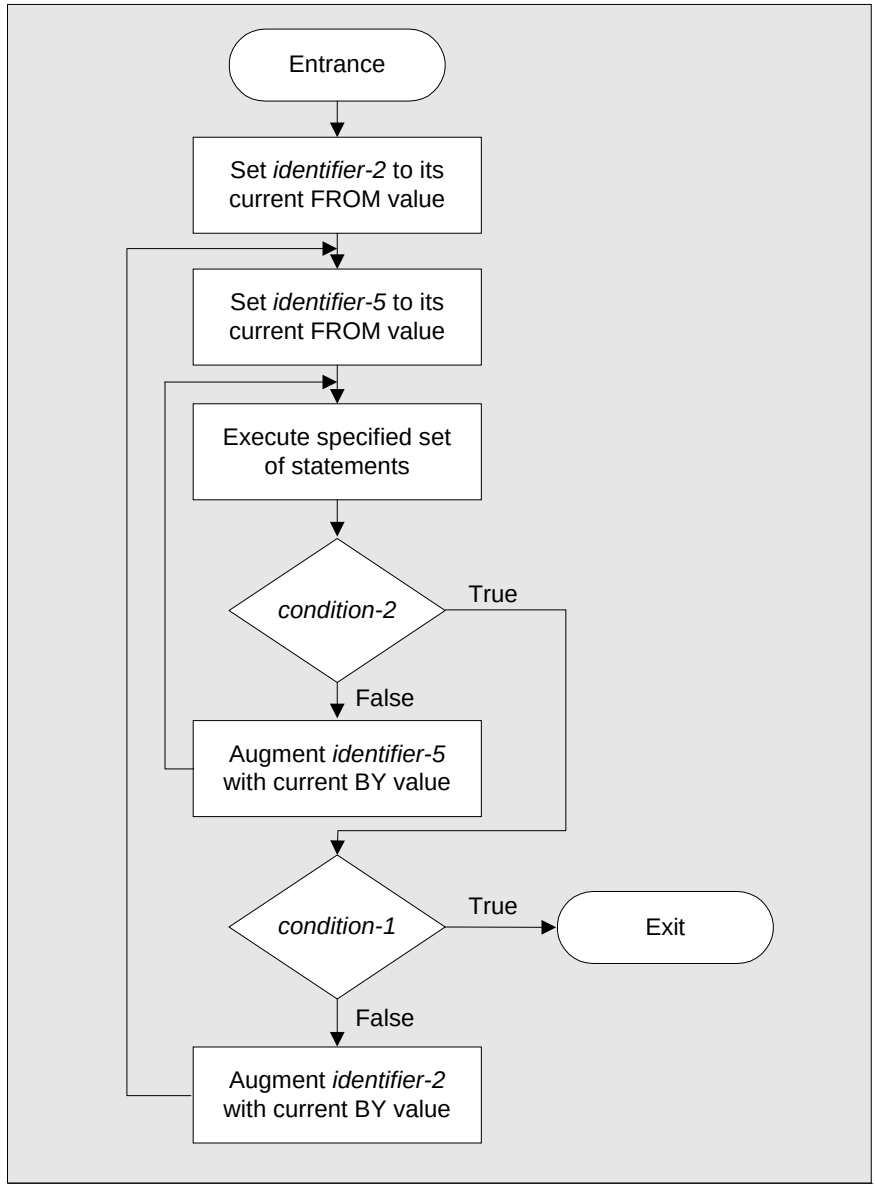
This PERFORM . . . VARYING statement has a TEST AFTER phrase and no AFTER phrase.

When control reaches the end of this form of the PERFORM . . . VARYING statement, *identifier-2* contains the same value it contained at the end of the most recent execution of the specified set of statements.

For a PERFORM . . . VARYING statement that has one AFTER phrase and a TEST AFTER phrase, the data item referred to by *identifier-2* is set to *literal-1* or the current value of the data item referred to by *identifier-3*, then the data item referred to by *identifier-5* is set to *literal-3* or the current value of the data item referred to by *identifier-6*. The specified set of statements is executed once and *condition-2* is tested. If it is false, the data item referred to by *identifier-5* is augmented by *literal-4* or the current value of the data item referred to by *identifier-7* and the specified set of statements is reexecuted with subsequent reevaluation of *condition-2*. This inner cycle of execution, testing and augmentation continues until *condition-2* is found to be true, at which time *condition-1* is tested. If it is true, control is transferred to the end of the PERFORM statement. If it is false, the data item referred to by *identifier-2* is augmented by *literal-2* or the current value of the data item referred to by *identifier-4*. *identifier-5* is set to *literal-3* or the current value of the data item referred to by *identifier-6*; and the specified set of statements is reexecuted with subsequent reevaluation of *condition-2*. This outer cycle continues until both *condition-1* and *condition-2* are found to be true.

Figure 6-4 represents this sequence of actions.

Figure 6-4: PERFORM . . . VARYING Statement



This PERFORM . . . VARYING statement has one AFTER phrase and a TEST AFTER phrase.

When control reaches the end of this form of the PERFORM . . . VARYING 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.

The preceding definition of the operation of a Format 4 PERFORM statement complies with ANSI COBOL 1985. It should be noted that this definition differs slightly from the definition in ANSI COBOL 1974, with which earlier versions of RM/COBOL complied. The difference is in the point at which inner cycle loop variables are reset to their FROM values.

It can have an effect only on Format 4 PERFORM statements that specify one or more AFTER phrases and that specify an inner FROM operand that is dependent on one of the higher-level loop operands. Most Format 4 PERFORM statements are not of this form, and are, therefore, not affected by this change. In situations where it is necessary to preserve compatibility with earlier versions of COBOL in this regard, two courses of action are possible: either modify the text of the source program, replacing the Format 4 PERFORM statement with an appropriate sequence of IF, MOVE, ADD and Format 1 PERFORM statements; or make use of the Compile Command option that causes the RM/COBOL compiler to treat Format 4 PERFORM statements as before. The *RM/COBOL User's Guide* contains further information on this option and the language features it controls.

During the execution of the specified set of statements associated with the PERFORM statement, any change to the VARYING variable (the data item referred to by *identifier-2* and *index-name-1*), the BY variable (the data item referred to by *identifier-4*), the AFTER variable (the data item referred to by *identifier-5* and *index-name-3*), or the FROM variable (the data item referred to by *identifier-3* and *index-name-2*) are taken into consideration and affect the operation of the PERFORM statement.

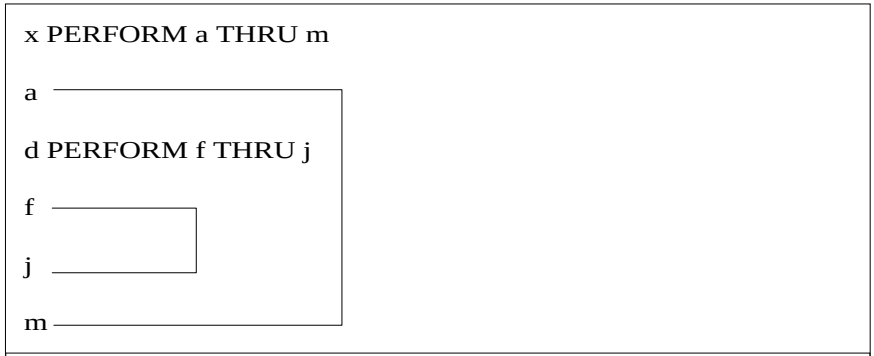
When the data items referred to by two identifiers are varied, the data item referred to by *identifier-5* goes through a complete cycle (FROM, BY, UNTIL) each time the contents of the data item referred to by *identifier-2* are varied. When the contents of three or more data items referred to by identifiers 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.

The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the implicit transfer of control to the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control 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.

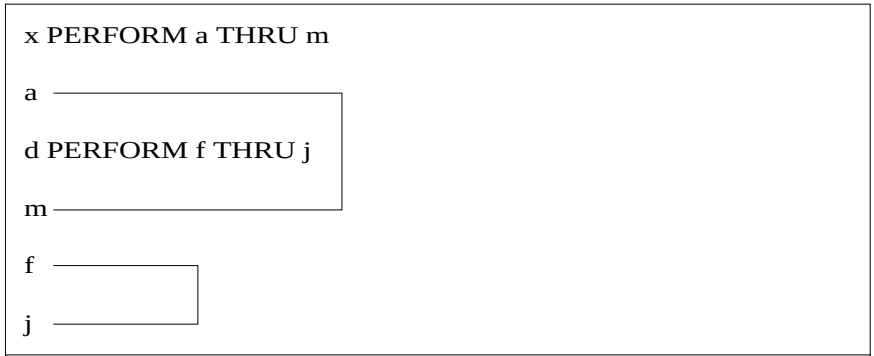
If the specified set of statements for one PERFORM statement includes another PERFORM statement, the specified set of statements associated with the inner PERFORM must itself be either totally included in, or totally excluded from, the logical sequence referred to by the outer PERFORM statement. Thus an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit. This is illustrated in Figure 6-5, Figure 6-6, and Figure 6-7.

Figure 6-5: PERFORM Statement Examples



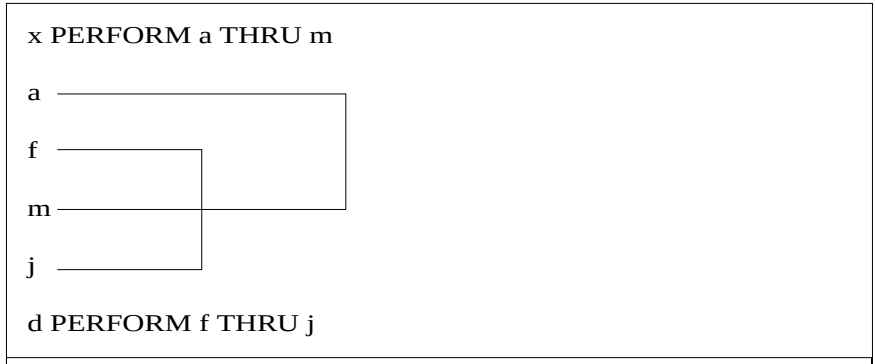
Performed statements totally included in logical sequence referred to by first PERFORM.

Figure 6-6: PERFORM Statement Examples



Performed statements totally excluded from logical sequence referred to by first PERFORM.

Figure 6-7: PERFORM Statement Examples



Second PERFORM excluded in logical sequence referred to by first PERFORM.

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, paragraphs, or both, wholly contained in one or more nonindependent segments.
- Sections, paragraphs, or both, 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, paragraphs, or both, wholly contained in one or more nonindependent segments.
- Sections, paragraphs, or both, wholly contained in the same independent segment as the PERFORM statement.

PERFORM Statement Examples

```
PERFORM INTIALIZATION-PROCEDURE.


PERFORM GROUP1 THROUGH GROUP5.


PERFORM
  DISPLAY "Ending run unit now"
  STOP RUN
END-PERFORM.


PERFORM STEP-UP COUNT-1 TIMES.


PERFORM 4 TIMES
  ADD ITEM-COUNT TO ITEM-COUNT
END-PERFORM.


SET EOF TO FALSE.
PERFORM UNTIL EOF
  READ INPUT-FILE
  AT END SET EOF TO TRUE
  NOT AT END ADD 1 TO RECORD-COUNT
  END-READ
END-PERFORM.


PERFORM ITEM-PROCEDURE
  WITH TEST AFTER UNTIL ITEM-COUNT = 0.


PERFORM VARYING T1-IX FROM 1 BY 1
                   UNTIL T1-IX > 100
  DISPLAY E1-FIELD(T1-IX)
    LINE  E1-LINE(T1-IX)
    COL   E1-COL(T1-IX)
END-PERFORM.


PERFORM TABLE-INITIALIZE
  VARYING IX1 FROM 1 BY 1 UNTIL IX1 > 5
  AFTER   IX2 FROM 1 BY 1 UNTIL IX2 > 10.
```

PURGE Statement

The PURGE statement eliminates from the Message Control System (MCS) a partial message which has been released by one or more SEND statements.

PURGE *cd-name-1*

cd-name-1 must reference an output CD or an input-output CD.

Execution of a PURGE statement causes the MCS to eliminate any partial message awaiting transmission to the destinations specified in the CD referred to by *cd-name-1*.

Any message that has associated with it an EMI or EGI is not affected by the execution of a PURGE statement.

The contents of the status key data item and the contents of the error key data item (if applicable) of the area referenced by *cd-name-1* are updated by the MCS.

PURGE Statement Examples

```
PURGE COM-LINE-1.

PURGE COM-LINE-2.
```

READ Statement

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.

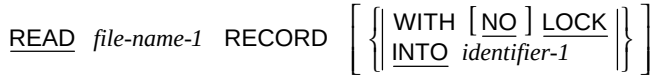
Format 1: Read Sequential Access

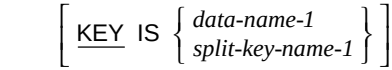
[AT END *imperative-statement-1*]

[NOT AT END *imperative-statement-2*]

[END-READ]

Format 2: Read Random Access





[INVALID KEY *imperative-statement-1*]

[NOT INVALID KEY *imperative-statement-2*]

[END-READ]

The file referenced by *file-name-1* must be open in the INPUT or I–O mode at the time this statement is executed.

In a Format 1 READ statement, the NEXT phrase causes the next logical record to be retrieved from the file, and the PREVIOUS phrase causes the previous logical record to be retrieved. The PREVIOUS phrase may not be specified for a sequential organization file.

For a file in which sequential access mode is specified, a Format 1 READ statement must be used. If both the NEXT phrase and the PREVIOUS phrase are omitted from a Format 1 READ statement for a file in sequential access mode, the default is NEXT.

For a file in which dynamic access mode is specified and records are to be retrieved sequentially using Format 1 READ statements, either the NEXT phrase or the PREVIOUS phrase must be specified.

Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for *file-name-1*.

The KEY phrase may be specified only when the organization of *file-name-1* is indexed. When the KEY phrase is present, *data-name-1* or *split-key-name-1* must be the name of one of the record keys associated with *file-name-1*. *data-name-1* or *split-key-name-1* may be qualified.

The setting of the file position indicator at the start of the execution of a Format 1 READ statement is used in determining the record to be made available according to the following rules. Comparisons for records in sequential files relate to the record number. Comparisons for records in relative files relate to the relative key number. Comparisons for records in indexed files relate to the value of the current key of reference, and the comparisons of key values are made according to the collating sequence of the file.

1. If the file position indicator indicates that no valid next record has been established, execution of the READ statement is unsuccessful.

2. If the file position indicator indicates that an optional input file is not present, execution proceeds as described below for the case when no next record exists.

3. If the file position indicator was established by a previous OPEN or START statement, and

   a. PREVIOUS was not specified, the first existing record in the file whose record number or key value is greater than or equal to the file position indicator is selected.

   b. PREVIOUS was specified, the first existing record in the file whose record number or key value is less than or equal to the file position indicator is selected.

4. If the file position indicator was established by a previous READ statement, and the file is a sequential or relative file, or an indexed file whose current key of reference does not allow duplicates, and

   a. PREVIOUS was not specified, the first existing record in the file whose record number or key value is greater than the file position indicator is selected.

   b. PREVIOUS was specified, the first existing record in the file whose record number or key value is less than the file position indicator is selected.

5. For indexed files, if the file position indicator was established by a previous READ statement, and the current key of reference does allow duplicates, and

   a. PREVIOUS was not specified, the first record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately after the record that was made available by that previous READ statement, or whose key value is greater than the file position indicator, is selected.

   b. PREVIOUS was specified, the first record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately before the record that was made available by that previous READ statement, or whose key value is less than the file position indicator, is selected.

If a record is found that satisfies these requirements, and there is no record lock conflict for that record, it is made available in the record area for the file.

If no record is found that satisfies these requirements, the file position indicator is set to indicate that no next record exists, and execution proceeds as described below for the case when no next record exists.

If a record is made available, the file position indicator is updated as follows:

- For sequential files, the file position indicator is set to the record number of the record made available.
- For relative files, the file position indicator is set to the relative record number of the record made available. If the RELATIVE KEY clause is specified for *file-name-1* and the number of significant digits in the relative record number of the selected record is larger than the size of the relative key data item, the file position indicator is set to indicate this condition and execution proceeds as described below for the case when no next record exists.
- For indexed files, the file position indicator is set to the value of the current key of reference of the record made available.

The execution of the READ statement causes the value of the file status data item, if any, associated with *file-name-1* to be updated.

When the logical records of a file are described with more than one record description, they share the same storage area; this is equivalent to an implicit redefinition of the area. If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for *file-name-1*, the portion of the record area that is to the right of the last valid character read is undefined. If the number of character positions in the record that is read is greater than the maximum size specified by the record description entries for *file-name-1*, the record is truncated on the right to the maximum size. In either case, the READ statement is successful and an I–O status value is set indicating that a record length conflict has occurred.

For a Format 1 READ statement, if the file position indicator indicates that no next (or previous) logical record exists, or that an optional input file is not present, the NOT AT END phrase, if specified, is ignored, and the following operations occur in the order specified:

1. An I–O status value is derived from the setting of the file position indicator and stored into the file status data item for the file, if there is one.

2. If the AT END phrase is specified in the Format 1 READ statement, control is transferred to the imperative statement in the AT END phrase. Any USE procedure associated with *file-name-1* is not executed.

3. If the AT END phrase is not specified, the applicable USE procedure, if there is one, is executed. Upon return from the USE procedure, control is transferred to the end of the READ statement. If there is no applicable USE procedure, an error message is produced and the run unit is terminated. The runtime can be configured, as described for the DEFAULT-USE-PROCEDURE keyword of the COMPILER-OPTIONS record in Chapter 10, *Configuration*, of the *RM/COBOL User's Guide*, to assume that a default empty USE procedure is applicable, thus causing execution to continue at the next executable statement after the READ statement.

If the at end condition occurs, execution of the Format 1 READ statement is unsuccessful. Following the unsuccessful execution of the READ statement, the contents of the associated record area are undefined and the file position indicator is set to indicate that no valid next record has been established. A further Format 1 READ statement for that file must not be executed without first executing one of the following:

- A successful CLOSE statement followed by the execution of a successful OPEN statement for that file
- A successful START statement for that file
- A successful Format 2 READ statement for that file

If an at end condition does not occur during the execution of a Format 1 READ statement, the AT END phrase and its associated imperative statement are ignored, if specified, and the following actions occur:

- The file position indicator is set and the I–O status value associated with the file-name is updated and stored into the file status data item for the file, if there is one.
- If an exception condition other than at end exists, control is transferred according to rules of the USE procedure applicable to the file-name.
- If no exception condition exists, the record is made available in the record area and any implicit move resulting from the presence of an INTO phrase is executed. Control is transferred to the end of the READ statement or to *imperative-statement-2*, if specified, in the NOT AT END phrase. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the READ statement.

For relative files if the RELATIVE KEY phrase is specified, the execution of a Format 1 READ statement updates the contents of the relative key data item such that it contains the relative record number of the record made available.

For relative files the execution of a Format 2 READ statement sets the file position indicator to, and makes available, the record whose relative record number is contained in the data item named in the RELATIVE KEY phrase for the file. If the file does not contain such a record, the invalid key condition exists and execution of the READ statement is unsuccessful.

For an indexed file being sequentially accessed using the NEXT phrase (specified either implicitly or explicitly) in a Format 1 READ statement, 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 that create such duplicate values. If the file is being sequentially accessed using the PREVIOUS phrase in a Format 1 READ statement, the records with duplicate keys are made available in reverse of the order that they are released or made duplicate.

In single record locking modes any record lock held by the run unit for *file-name-1* is released upon execution of the READ statement. The successful execution of the READ statement may obtain a record lock on the accessed record as described in the discussion of the LOCK phrase on page 6-152.

In multiple record locking modes any record locks held by the run unit for *file-name-1* are not released upon execution of the READ statement.

KEY Phrase

KEY IS { *data-name-1* / *split-key-name-1* }

For an indexed file if the KEY phrase is specified in a Format 2 READ statement, *data-name-1* or *split-key-name-1* is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different key of reference is established for the file.

If the KEY phrase is not specified in a Format 2 READ statement, 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 READ statements for the file until a different key of reference is established for the file.

For indexed files the execution of a Format 2 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 is positioned 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.

For relative files the KEY phrase cannot be specified.

LOCK Phrase

WITH [NO] LOCK

The LOCK phrase may be specified to control record locking for a shared input-output file. If the file is not a shared input-output file, the LOCK phrase is ignored and the execution of the READ statement does not attempt to obtain a lock on the record accessed. For a file open in the INPUT mode, the execution of the READ statement never attempts to obtain a lock on the record accessed.

In manual record locking modes the READ statement only attempts to lock the record accessed when the LOCK phrase, without the NO option, is specified. If the record accessed by the READ statement is to be subsequently rewritten or deleted, the LOCK phrase, without the NO option, should be specified in a READ statement executed in manual record locking modes. For a READ statement executed in manual record locking modes the NO LOCK phrase is redundant.

In automatic record locking modes the READ statement automatically attempts to lock the record accessed except when the NO LOCK phrase is specified. Specifying NO LOCK will reduce file contention in a shared file environment when the record accessed by the READ statement is not to be subsequently rewritten or deleted. In automatic record locking modes the LOCK phrase, without the NO option, is redundant.

When a READ statement attempts to obtain a record lock and the record accessed is already locked by another concurrently executing run unit, the subsequent action depends on the form of the program:

- If the program declares both a file status data item for *file-name-1* and an applicable USE procedure for *file-name-1*, the READ statement completes unsuccessfully with an I–O status value that indicates a locked record conflict and the USE procedure is performed. The status of the file position indicator in this case is described in the *RM/COBOL User's Guide*.

- If the conditions in the above paragraph are not satisfied, the runtime system waits for the other run unit to unlock the record before completing the READ statement execution for this run unit.

If the record is already locked by this run unit through another COBOL file-name that refers to the same physical file, the READ statement will not wait but will complete unsuccessfully regardless of whether both a file status data item and applicable USE procedure are defined in the program.

If the record is already locked by this run unit through *file-name-1*, the READ statement completes successfully and the lock status of the accessed record is not changed.

When a READ statement does not attempt to obtain a lock on the record accessed, the lock status of the record is not significant. The current contents of the record are obtained at the time of the execution of the READ statement without indication of its locked or unlocked status.

See “Record Locking” on page 5-69 for additional information on record locking.

INTO Phrase

INTO *identifier-1*

If the INTO phrase is specified, the record being read is moved from the record area to the area specified by *identifier-1* according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The size of the current record is determined by rules specified in the RECORD clause. If the file description entry contains a RECORD IS VARYING clause, the implied move is a group move. The implied move does not occur if the execution of the READ 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.

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-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 storage area associated with *identifier-1* and the record area associated with *file-name-1* must not be the same storage area.

INVALID KEY and NOT INVALID KEY Phrases

INVALID KEY *imperative-statement-1*

NOT INVALID KEY *imperative-statement-2*

The causes of the invalid key condition for a READ statement execution have been indicated in the preceding text. See the discussions of relative organization input-output and indexed organization input-output that begin on pages 5-50 and 5-59, respectively, for additional information on the invalid key condition and the use of the INVALID KEY clause.

READ Statement Examples

```
READ TRANSACTION-FILE RECORD.


READ LOG-FILE NEXT RECORD INTO RECORD-SAVE
AT END SET EOF TO TRUE
NOT AT END PERFORM PROCESS-LOG-RECORD
END-READ.


READ INVENTORY-FILE PREVIOUS RECORD WITH LOCK
AT END DISPLAY "Beginning-of-file reached."
END-READ.


READ DATA-BASE NEXT RECORD WITH NO LOCK
AT END PERFORM EOF-PROCEDURE.


READ INVENTORY-FILE RECORD
INVALID KEY PERFORM BAD-KEY-PROCEDURE
END-READ.


READ DATA-BASE WITH NO LOCK INTO RECORD-WORK-AREA
INVALID KEY DISPLAY "Bad key"
NOT INVALID KEY PERFORM PROCESS-WORK-AREA
END-READ.
```

RECEIVE Statement

The RECEIVE statement makes available to the program a message or a message segment and pertinent information about that data.

```
RECEIVE cd-name-1 { MESSAGE } INTO identifier-1
                  { SEGMENT }

    [ NO DATA imperative-statement-1 ]

    [ WITH DATA imperative-statement-2 ]

    [ END-RECEIVE ]
```

cd-name-1 must reference an input CD or an input-output CD.

If *cd-name-1* references an input CD, the contents of the data items specified by *data-name-1* (SYMBOLIC QUEUE) through *data-name-4* (SYMBOLIC SUB-QUEUE-3) of the area referenced by *cd-name-1* designate the queue structure containing the message.

If *cd-name-1* references an input-output CD, the contents of the data item specified by *data-name-3* (SYMBOLIC TERMINAL) of the area referenced by *cd-name-1* designates the source of the message.

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* aligned to the left without space fill.

The data items identified by *cd-name-1* are appropriately updated by the MCS at each execution of a RECEIVE statement.

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 return any portion of a message to the object program until the entire message is available to the MCS, even when the SEGMENT phrase of the RECEIVE statement is specified.

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.

NO DATA and WITH DATA Phrases

NO DATA *imperative-statement-1*

WITH DATA *imperative-statement-2*

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*. In the latter case, execution continues according to the rules for each statement 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 RECEIVE statement.

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 following actions occurs:

- 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 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.
- 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*.
- If one or more queues or subqueues are unknown to the MCS, the appropriate status key value is stored and control is then transferred as if data had been made available.

MESSAGE Phrase

MESSAGE

If the MESSAGE phrase is used, end of segment indicators are ignored, and the following rules apply to data transfer:

1. 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*.

2. 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.

3. If the 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 that specify the same queue structure. The remainder of the message is treated as a new message.

4. If an end of group indicator is associated with the text accessed by the RECEIVE statement, the existence of an end of message indicator is implied.

SEGMENT Phrase

SEGMENT

If the SEGMENT phrase is used, the following rules apply:

1. 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*.

2. If the 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.

3. 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 that specify the same queue structure. The remainder of the segment is treated as a new segment.

4. 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.

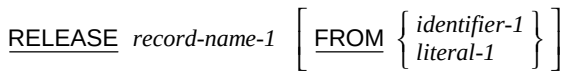
RECEIVE Statement Examples

```
RECEIVE COM-PORT MESSAGE INTO MESSAGE-BUFFER
NO DATA PERFORM NO-MESSAGE-PROCEDURE
WITH DATA PERFORM PROCESS-MESSAGE-PROCEDURE
END-RECEIVE.
```

```
RECEIVE COM-LINE-2 SEGMENT INTO SEGMENT-BUFFER
NO DATA MOVE
  DEFAULT-SEGMENT TO SEGMENT-BUFFER
END-RECEIVE.
```

RELEASE Statement

The RELEASE statement transfers records to the initial phase of a sort operation.



A RELEASE statement may be used only within the range of an input procedure associated with a SORT statement for a file whose sort-merge file description entry contains *record-name-1*.

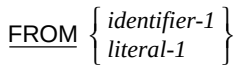
record-name-1 must be the name of a logical record in the associated sort-merge file description entry and may be qualified.

record-name-1 and *identifier-1* must not refer to the same storage area.

The execution of a RELEASE statement causes the record named by *record-name-1* to be released to the initial phase of a sort operation.

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.

FROM Phrase



If the FROM phrase is used, *literal-1* or the contents of *identifier-1* are moved to *record-name-1*, then the contents of *record-name-1* are released to the sort file. Moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING phrase (see page 6-116). The information in the record area is no longer available, but the information in the data area associated with *identifier-1* is available.

RELEASE Statement Example

```
SORT-INPUT-PROCEDURE.
    SET INPUT-EOF TO FALSE.
    OPEN INPUT INPUT-FILE.
    PERFORM UNTIL INPUT-EOF
      READ INPUT-FILE AT END
        SET INPUT-EOF TO TRUE
      NOT AT END
        RELEASE SORT-RECORD FROM INPUT-RECORD
      END-READ
    END-PERFORM.
    CLOSE INPUT-FILE.
```

RETURN Statement

The RETURN statement either sorted records from the final phase of a sort operation or merged records during a merge operation.

```
RETURN file-name-1 RECORD [INTO identifier-1]
    [AT END imperative-statement-1]
    [NOT AT END imperative-statement-2]
    [END-RETURN]
```

file-name-1 must be described by a sort-merge file description entry in the Data Division.

A RETURN statement may be used only within the range of an output procedure associated with a SORT or MERGE statement for *file-name-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 storage area associated with *identifier-1* and the record area associated with *file-name-1* must not be the same storage area.

When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the RETURN statement.

The execution of the RETURN statement causes the next record, in the order specified by the keys listed in the SORT or MERGE statement, to be made available for processing in the record area associated with *file-name-1*.

If no next record exists in the file referenced by *file-name-1*, the at end condition exists and control is transferred to *imperative-statement-1* in the AT END phrase. Execution continues according to the rules for each statement in *imperative-statement-1*. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the RETURN statement and the NOT AT END phrase, if specified, 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-1* are undefined. After the

execution of *imperative-statement-1* in the AT END phrase, no further RETURN statements may be executed as part of the current output procedure.

If the at end condition does not occur during the execution of a RETURN statement, then after the record is made available and after executing any implicit move resulting from the presence of an INTO phrase, control is transferred to *imperative-statement-2* in the NOT AT END phrase, if specified; otherwise, control is transferred to the end of the RETURN statement.

If the INTO phrase is specified, the current record is moved from the input area to the area specified by *identifier-1* according to the rules for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if there is an at end condition. Any subscripting associated with *identifier-1* is evaluated after the record has been returned and immediately before it is moved to the data item.

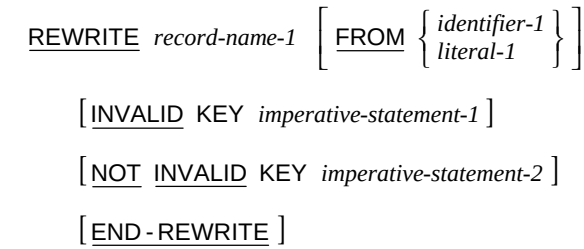
When the INTO phrase is used, the data is available in both the input record area and the data area associated with *identifier-1*.

RETURN Statement Example

```
SORT-MERGE-OUTPUT-PROCEDURE.
    OPEN OUTPUT OUTPUT-FILE.
    SET SORT-EOF TO FALSE.
    PERFORM UNTIL SORT-EOF
      RETURN SORT-FILE RECORD INTO OUTPUT-RECORD
      AT END SET SORT-EOF TO TRUE
      NOT AT END
        WRITE OUTPUT-RECORD
      END-RETURN
    END-PERFORM.
    CLOSE OUTPUT-FILE.
```

REWRITE Statement

The REWRITE statement logically replaces a record existing in a mass storage file.



record-name-1 and *identifier-1* must not refer to the same storage area.

record-name-1 is the name of a logical record in the File Section of the Data Division and may be qualified.

The file associated with *record-name-1* must be a mass storage file and must be open in the I-O mode at the time of execution of this statement.

The INVALID KEY and the NOT INVALID KEY phrases must not be specified for a REWRITE statement which references a sequential file or a relative file in sequential access mode.

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.

For indexed files the INVALID KEY phrase must be specified in the REWRITE statement for files for which an appropriate USE procedure is not specified.

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 runtime system replaces the record that was accessed by that READ statement.

When an indexed file is described with the DUPLICATES phrase in the RECORD KEY clause, the REWRITE statement in the dynamic access mode is executed as if the file were in the sequential access mode and the REWRITE statement in the random access mode is not allowed.

The file position indicator is not affected by the execution of a REWRITE statement.

The execution of the REWRITE statement causes the value of the file status data item, if any, associated with the file to be updated.

The record to be replaced by the execution of the REWRITE statement must not be locked by another run unit. For a shared input-output file, the run unit executing the REWRITE statement should obtain a record lock by preceding the REWRITE statement with a READ statement that locks the record to be replaced. If the run unit does not already hold a lock on the record to be replaced, the runtime system will attempt to obtain the lock. If the lock cannot be obtained because another run unit holds a lock on the record, subsequent action of the program is as described for the READ statement when attempting to lock a record already locked by another run unit. If the lock cannot be obtained because this run unit holds a lock on the record through another COBOL file-name, the REWRITE statement is unsuccessful. See “File Locking” on page 5-68 and “Record Locking” on page 5-69 for additional information on coordinating file updates in a shared file environment.

In single record locking modes, any record lock held by the run unit for the file associated with *record-name-1* is released after execution of the REWRITE statement.

In multiple record locking modes, record locks are not released except for the record lock obtained by the runtime system when the record to be replaced was not locked by the run unit prior to execution of the REWRITE statement.

For a relative file accessed in a random or dynamic access mode, the runtime system replaces the record specified by the contents of the relative key data item of the file. If the file does not contain the record selected by that key value, the invalid key condition exists.

For an indexed file accessed in the sequential access mode, the record to be replaced is selected by the value of the prime record key. When the REWRITE statement is executed, the value of the prime record key of the record to be replaced must be equal to the value of the prime record key of the last record read from the file. When the DUPLICATES phrase is specified in the RECORD KEY clause of the file control entry for the file, the record to be replaced is the one accessed by the previously executed READ statement.

For an indexed file in the random or dynamic access mode, the record to be replaced is selected by the prime record key.

For an indexed file, execution of the REWRITE statement for a record that has an alternate record key occurs as follows:

- When the value of a specific alternate record key is not changed, the order of retrieval when that key is the key of reference remains unchanged.
- When the value of a specific alternate record key is changed, the subsequent order of retrieval of that record may be changed when that specific alternate record key is the key of reference. When duplicate key values are permitted, the record is logically positioned last within the set of duplicate records containing the same alternate record key value as the alternate record key value that was placed in the record.

For indexed files the invalid key condition exists under any of the following circumstances:

- When the access mode of the file is sequential and the value of the prime record key of the record to be replaced is not equal to the value of the prime record key of the last record read from the file.
- When the access mode of the file is dynamic or random and the value of the prime record key of the record to be replaced is not equal to the value of the prime record key of any record existing in the file.
- When the value of an alternate record key of the record to be replaced, for which duplicates are not allowed, equals the value of the corresponding data item of a record already existing in the file.

When the invalid key condition is recognized for both relative and indexed files, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the contents of the record area are unaffected, and the I-O status value of the file associated with *record-name-1* is set to a value indicating the cause of the condition.

Transfer of control following the successful or unsuccessful execution of the REWRITE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the REWRITE statement. See the discussion of invalid key conditions pages 5-56 and 5-65.

See the discussions of relative organization input-output and indexed organization input-output that begin on pages 5-50 and 5-59, respectively, for additional information on the invalid key condition and the use of the INVALID KEY phrase.

For sequential files, if the number of character positions in the record referenced by *record-name-1* is not equal to the number of character positions in the record being replaced, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the contents of the record area are unaffected, and the I-O

status value of the file associated with *record-name-1* is set to a value indicating the cause of the condition.

For relative and indexed files, the number of character positions in the record referenced by *record-name-1* need not be the same as the number of character positions in the record being replaced. However, if it is larger than the largest or smaller than the smallest number of character positions allowed by the RECORD IS VARYING clause associated with the file, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the contents of the record area are unaffected, and the I-O status value of the file associated with *record-name-1* is set to a value indicating the cause of the condition.

FROM Phrase

FROM { *identifier-1* / *literal-1* }

The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of a move from *identifier-1* or *literal-1* to *record-name-1* 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.

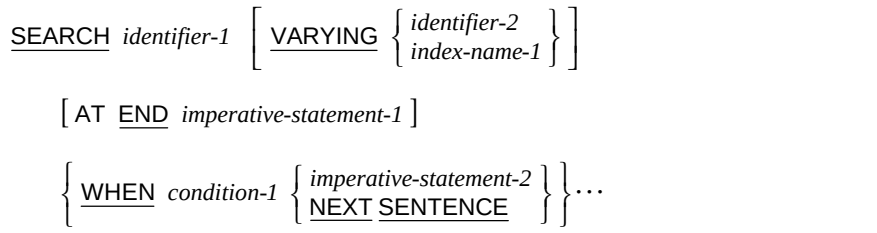
REWRITE Statement Examples

```
REWRITE LOG-RECORD OF LOG-FILE.


REWRITE LOG-RECORD FROM "END-OF-BATCH"
END-REWRITE.


REWRITE INVENTORY-RECORD
INVALID KEY PERFORM INVALID-KEY-HANDLER
END-REWRITE.


REWRITE DB-RECORD OF DATA-BASE
INVALID KEY
  REWRITE INVENTORY-RECORD END-REWRITE
END-REWRITE.
```

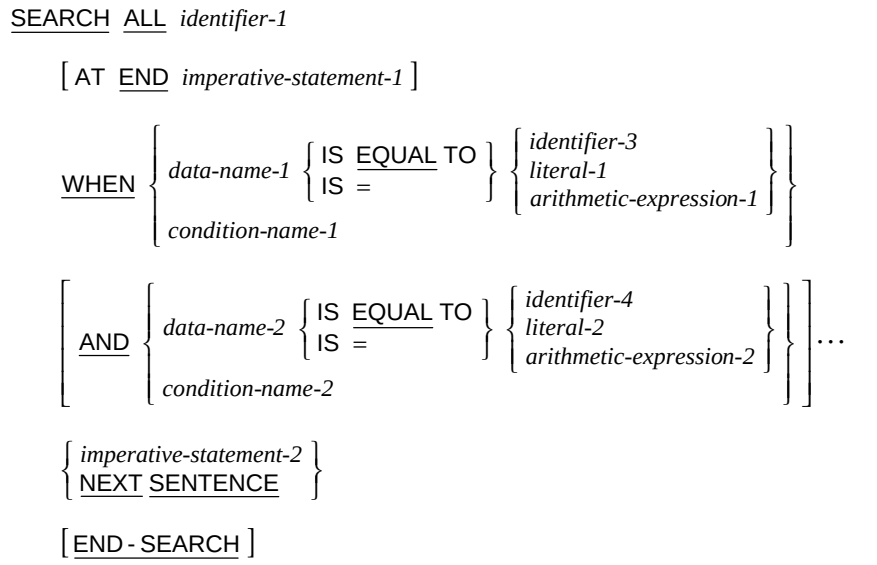
SEARCH Statement

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index-name to indicate that table element.

Format 1: Search (Serial)



Format 2: Search All (Binary)



In both Formats 1 and 2, *identifier-1* must not be subscripted or reference modified, but its description must contain an OCCURS clause with an INDEXED BY phrase. The description of *identifier-1* in Format 2 must also contain the KEY IS phrase in its OCCURS clause.

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.

In Format 1, *condition-1* may be any conditional expression.

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 phrase of the OCCURS clause of *identifier-1*. Each data-name may be qualified. Each data-name must be subscripted by the first index-name associated with *identifier-1* along with other indexes or literals as required, and must be referenced in the KEY phrase of the OCCURS clause of *identifier-1*. *identifier-3*, *identifier-4*, or identifiers specified in *arithmetic-expr-1* or *arithmetic-expr-2* must not be referenced in the KEY phrase of the OCCURS clause of *identifier-1* or be subscripted by the first index-name associated with *identifier-1*.

In Format 2, when multiple keys are defined and a data-name in the KEY phrase of the OCCURS clause of *identifier-1* is referenced, or when a condition-name associated with a data-name in the KEY phrase of the OCCURS clause of *identifier-1* is referenced, all preceding data-names in the KEY phrase of the OCCURS clause of *identifier-1* or their associated condition-names must also be referenced.

General Rules

The following general rules apply to the SEARCH statement:

1. If Format 1 is used, a serial type of search operation takes place, starting at the current index setting.

    a. If, at the start of the execution of the search, the index-name associated with *identifier-1* contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for *identifier-1*, the search is terminated immediately. Then, if the AT END phrase is specified, *imperative-statement-1* is executed; if the AT END phrase is not specified, control passes to the end of the SEARCH statement.

    b. If, at the start of the execution of the search, the index-name associated with *identifier-1* contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for *identifier-1*, the SEARCH statement operates by evaluating the conditions in the WHEN phrases in the order in which they are written, making use of the index settings to

determine the occurrence of those items to be tested. If none of the conditions are satisfied, the index-name for *identifier-1* is incremented, and the process is repeated unless the index is out of range (in which case the search terminates as indicated in rule 1a). If one of the conditions is satisfied, the search terminates immediately and control passes to the imperative statement associated with that condition, if present, or if the NEXT SENTENCE phrase is associated with that condition, the control passes to the next executable sentence.

The index-name remains set at the value which causes the condition to be satisfied.

2. In a Format 2 search, the results of the SEARCH ALL operation are predictable only when both of these conditions exist:

   a. The data in the table is ordered in the same manner as described in the ASCENDING/DESCENDING KEY clause associated with the description of *identifier-1*.

   b. The contents of the key (or keys) referenced in the WHEN clause are sufficient to identify a unique table element.

3. If Format 2 of the SEARCH statement is used, a binary search technique is applied. The value of the index-name for *identifier-1* is varied in alternating directions and in progressively smaller steps until either a value is found for which all the conditions of the WHEN phrase are satisfied or it is determined that no value allows all of the conditions to be satisfied. In the latter case, control is passed to *imperative-statement-1* in the AT END phrase, if specified, or to the end of the SEARCH statement if there is no AT END phrase; in either case, the final setting of the index is not predictable. If a setting of the index is found for which all of the conditions are satisfied, control passes to *imperative-statement-2*, if specified, or if the NEXT SENTENCE phrase is specified, to the next executable sentence; in either case, the final setting of the index is the one for which the conditions are all satisfied. Regardless of the outcome of the SEARCH statement, the initial setting of the index is not significant.

4. After execution of *imperative-statement-1*, *imperative-statement-2*, and so forth, that does not contain an explicit transfer of control, control passes to the end of the SEARCH statement.

5. In Format 2, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY phrase of *identifier-1*. Any other index-names for *identifier-1* remain unchanged.

6. In Format 1, if the VARYING phrase is not used, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY phrase of *identifier-1*. Any other index-names for *identifier-1* remain unchanged.

7. In Format 1, if the VARYING *index-name-1* phrase is specified, and if *index-name-1* appears in the INDEXED BY phrase in the OCCURS clause referenced by *identifier-1*, 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 (or only) index-name given in the INDEXED BY phrase in the OCCURS clause referenced by *identifier-1* is used for the search. In addition, the following operations occur:

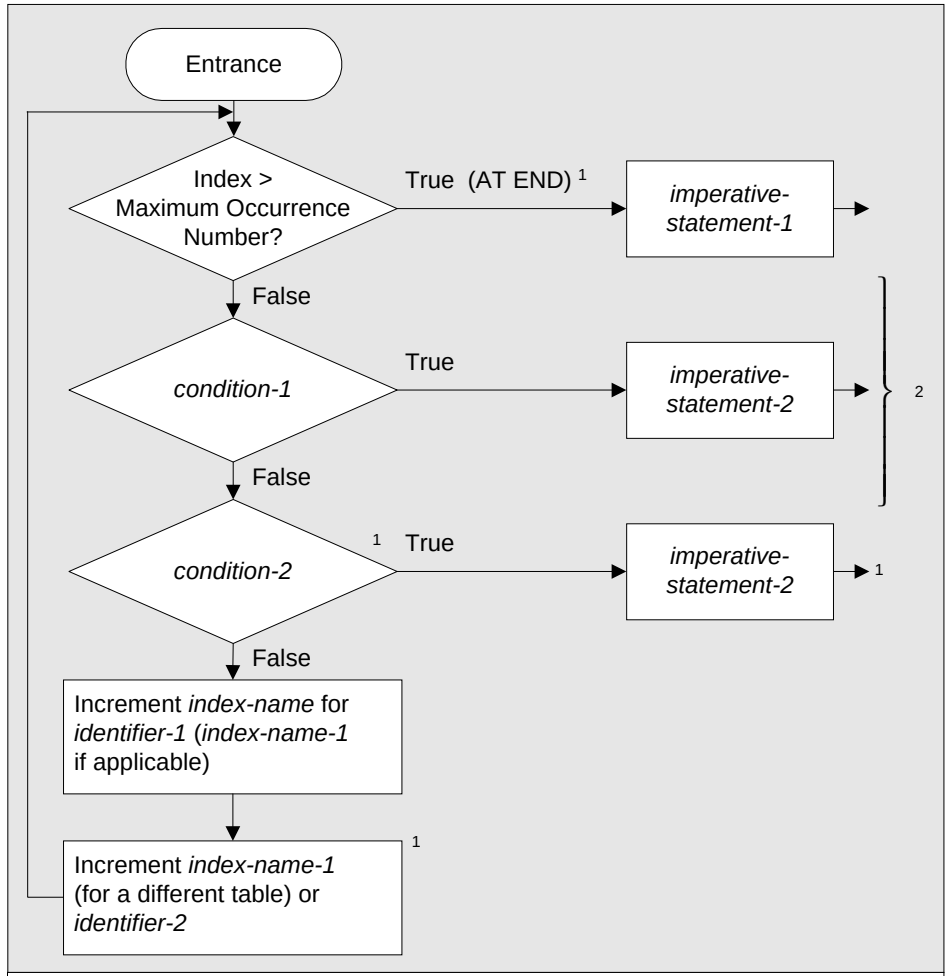
    a. If the VARYING *index-name-1* phrase is used, and if *index-name-1* appears in the INDEXED BY phrase in the OCCURS clause referenced by another table entry, the occurrence number represented by *index-name-1* is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with *identifier-1* is incremented.

    b. If the VARYING *identifier-2* phrase is specified, and *identifier-2* is an index data item, the data item referenced by *identifier-2* is incremented by the same amount as, and at the same time as, the index associated with *identifier-1* is incremented. 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 associated with *identifier-1* is incremented.

8. If *identifier-1* references a data item subordinate to a data item that contains an OCCURS clause, 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*, if present) is modified by the execution of the SEARCH statement. To search a multidimensional table it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

A representation of the action of a Format 1 SEARCH statement containing two WHEN phrases is shown in Figure 6-8.

Figure 6-8: SEARCH Statement



1 *These operations are options included* only *when specified in the SEARCH statement.*
2 *Each of these control transfers is to the end of the SEARCH statement unless the imperative statement contains an explicit transfer.*

This Format 1 SEARCH statement contains two WHEN phrases.

SEARCH Statement Examples

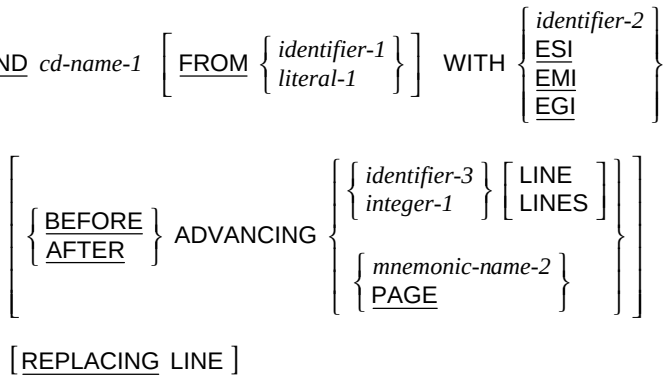
```
ACCEPT INPUT-NAME TAB PROMPT.
SET IX1 TO 1.
SEARCH STATE-NAME-TABLE VARYING IX1
AT END
  DISPLAY "The name """ INPUT-NAME
    """ is not in the state name table."
WHEN STATE-NAME(IX1) = INPUT-NAME
  PERFORM SETUP-BUFFERS   *> Note: uses current IX1 setting.
  DISPLAY "The abbreviation for the state of """
    STATE-BUFFER(1:STATE-COUNT)
    """ is """ STATE-ABBREV(IX1) ""","
    "and the capital is """ COL 5
    CAPITAL-BUFFER
WHEN STATE-CAPITAL(IX1) = INPUT-NAME
  PERFORM SETUP-BUFFERS   *> Note: uses current IX1 setting.
  DISPLAY
    "The city """ CAPITAL-BUFFER(1:CAPITAL-COUNT)
    " is the state capital of """
    STATE-BUFFER(1:STATE-COUNT) """."
WHEN STATE-ABBREV(IX1) = INPUT-NAME
  PERFORM SETUP-BUFFERS   *> Note: uses current IX1 setting.
  DISPLAY "The abbreviation """ STATE-ABBREV(IX1)
  """ stands for the state of """
  STATE-BUFFER(1:STATE-COUNT) ""","
  " and the state capital is """ COL 5 CAPITAL-BUFFER
END-SEARCH.



ACCEPT CURR-ABBREV TAB PROMPT.
SEARCH ALL STATE-NAME-TABLE
AT END
  DISPLAY "The abbreviation """ CURR-ABBREV
  """ is not in the state name table."
WHEN STATE-ABBREV(IX1) = CURR-ABBREV
  PERFORM SETUP-BUFFERS   *> Note: uses current IX1 setting.
  DISPLAY "The abbreviation """ STATE-ABBREV(IX1)
  """ stands for the state of """
  STATE-BUFFER(1:STATE-COUNT) ""","
  " and the state capital is """ COL 5 CAPITAL-BUFFER
END-SEARCH.
```

SEND Statement

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 (MCS).

Format 1: Send (Simple)

Format 2: Send (Advancing/Replacing)



cd-name-1 must reference an output CD or an input-output CD.

identifier-2 must reference a one-character integer without an operational sign.

When *identifier-3* is used in the ADVANCING phrase, it must be the name of an elementary integer item.

When the mnemonic-name phrase is used, the name must be identified with a feature-name that is a channel-name. See the syntax and rules for *mnemonic-name-2* that begin on page 3-9.

integer-1 or the value of the data item referenced by *identifier-3* may be zero.

General Rules

The following general rules apply to both formats of the SEND statement:

1. When a receiving communication device (printer, display screen, card punch, and so forth) is oriented to a fixed line size:

   a. Each message or message segment begins 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 so as to appear space-filled to the right.

   c. Excess characters of a message or message segment are not truncated. Characters are packed to a size equal to that of the physical line and then output 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, and so forth) is oriented to handle variable-length messages, each message or message segment begins on the next available character position of the communication device.

3. As part of the execution of a SEND statement, the MCS interprets the content of the text length data item of the area referenced by *cd-name-1* to be the user's indication of the number of leftmost character positions of the data item referenced by *identifier-1* from which data is to be transferred.

   If the content of the text length data item of the area referenced by *cd-name-1* is zero, no characters of the data item referenced by *identifier-1* are transferred.

   If the content of the text length data item of the area referenced by *cd-name-1* is 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 status key data item of the area referenced by *cd-name-1*, and no data is transferred.

4. As part of the execution of a SEND statement, the content of the status key data item of the area referenced by *cd-name-1* is updated by the MCS.

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 represented by Format 1 releases only a single portion of a message segment or a single portion of a message to the MCS. A single execution of a SEND statement represented by Format 2 never releases to the MCS more than a single message or a single message segment as indicated by the content of the data item referenced by *identifier-2* or by the specified indicator ESI, EMI or EGI.

   However, the MCS does not transmit any portion of a message to a communication device until the entire message has been released to the MCS.

7. During the execution of the run unit, the disposition of a portion of a message which is 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 hence 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 executions of SEND statements in the same run unit can cause the remaining portion of the message to be released.

For Format 2:

9. The content of the data item referenced by *identifier-2* indicates that the content of the data item referenced by *identifier-1*, when specified, is to have an associated end of segment indicator, end of message indicator, end of group indicator or no indicator (which implies a portion of a message or a portion of a segment). If *identifier-1* is not specified, only the indicator is transmitted to the MCS. See Table 6-9.

   Any character other than '1', '2' or '3' is interpreted as '0'.

   If the content of the data item referenced by *identifier-2* is other than '1', '2' or '3', and *identifier-1* is not specified, 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-1*, and no data is transferred.

Table 6-9: Data Item Contents

If the content of the data item referenced by *identifier-2* is	**then the content of data item referenced by *identifier-1* has associated with it**	**which means**
'0'	No indicator.	No indicator.
'1'	ESI	End of segment.
'2'	EMI	End of message.
'3'	EGI	End of group.

10. The WITH EGI phrase indicates to the MCS that the group of messages is complete.

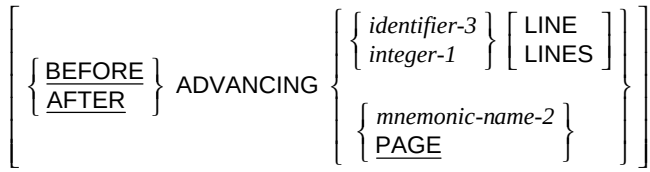
    The WITH EMI phrase indicates to the MCS that the message is complete.

    The WITH ESI phrase indicates to the MCS that the message segment is complete.

    The MCS recognizes these indications and uses them to maintain segment, message and group control.

11. The hierarchy of ending indicators is EGI, EMI and ESI. An EGI need not be preceded by ESI or EMI. An EMI need not be preceded by an ESI.

ADVANCING Phrase



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 ADVANCING phrase is ignored.

If *identifier-2* is specified and the content of the data item referenced by *identifier-2* is zero, the ADVANCING phrase is ignored.

On a device where vertical positioning is applicable and the ADVANCING phrase is not specified, the default advance is one line.

If vertical positioning is applicable, the following rules apply to the ADVANCING phrase:

1. If *identifier-3* or *integer-1* is specified, characters transmitted to the communication device are repositioned vertically downward the number of lines equal to the value associated with the data item referenced by *identifier-3* or *integer-1*.

2. If *mnemonic-name-2* is specified, characters transmitted to the communication device are positioned downward to the next occurrence of the channel indicator for the channel number associated with *mnemonic-name-2*. If the communication device does not support channel skipping, advancing defaults to ADVANCING 1 LINE.

3. If the BEFORE phrase is used, the message or message segment is represented on the communication device before vertical repositioning.

4. If the AFTER phrase is used, the message or message segment is represented on the communication device after vertical repositioning.

5. If PAGE is specified, characters transmitted to the communication device are represented before or after (depending on the phrase used) the device is repositioned to the next page. If PAGE is specified but has no meaning with a specific device, advancing defaults to ADVANCING 1 LINE.

6. When the receiving communication device is a character imaging device on which it is possible to present more than one character at the same position, and the device permits the choice of either the second or subsequent characters appearing superimposed on characters already displayed at that position or each character appearing in place of the characters previously transmitted to that line, then:

   a. If 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.

   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.

7. When the receiving communication device does not support the replacement of characters, regardless of whether the REPLACING phrase is 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.

8. When the receiving communication device does not support the superimposition of more than one character at the same position, regardless of whether 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.

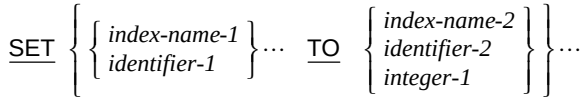
SEND Statement Examples

```
SEND COM-LINE-1 FROM "Enter your PIN: ".


SEND COM-LINE-2 FROM SEGMENT-BUFFER WITH ESI
  AFTER ADVANCING 3 LINES.
```

SET Statement

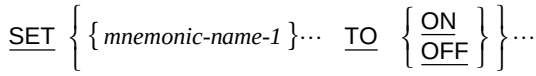
The SET statement is used to establish reference points for table handling operations, alter the status of external switches, and alter the value of conditional variables.

Format 1: Set Index

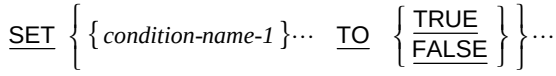


Format 2: Set Index Up/Down

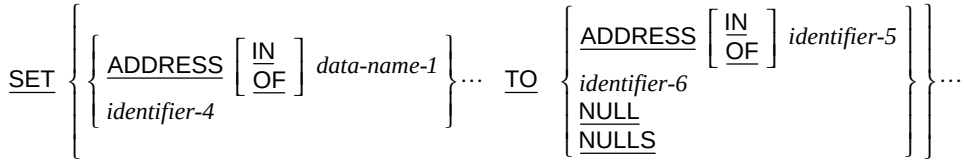
Format 3: Set Switch On/Off



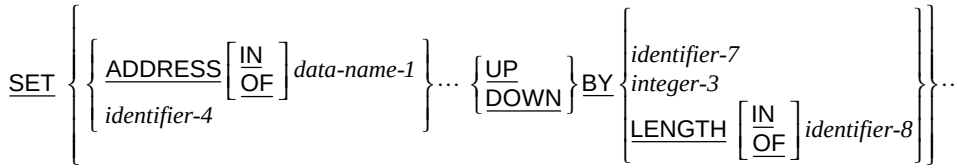
Format 4: Set Condition-Name True/False



Format 5: Set Pointer



Format 6: Set Pointer Up/Down



identifier-1 and *identifier-2* must name either index data items, or elementary items described as an integer.

identifier-3 and *identifier-7* must refer to elementary data items described as an integer.

integer-1 and *integer-2* may be signed. *integer-1* must be positive.

mnemonic-name-1 must be identified in the SPECIAL-NAMES paragraph of the Environment Division as one of the permissible switch-names SWITCH-1, SWITCH-2, . . ., SWITCH-8, or UPSI-0, UPSI-1, . . ., UPSI-7.

condition-name-1 must be associated with a conditional variable.

data-name-1 must be the name of a level 01 or level 77 data description entry that is described in the Linkage Section.

identifier-4 and *identifier-6* must refer to elementary data items described with POINTER usage.

In Format 4, if the TRUE phrase is specified, the Format 2 VALUE clause, which is described on page 4-58, for the condition-name must either not specify a relational operator prior to the first listed literal or that first relational operator must be one that includes an equality relation. See “Condition-Name Rules (Format 2 VALUE Clause)” on page 4-61 for additional information regarding the use of a relational operator in the VALUE clause and the existence of a true value for purposes of the SET statement.

In Format 4, if the FALSE phrase is specified, the FALSE phrase must be specified in the VALUE clause of the data description entry for *condition-name-1*.

General Rules

The general rules that apply to the SET statement are as follows:

Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

If *index-name-2* 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 associated table.

If *index-name-3* 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 associated 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 associated table. The value of the index associated with an index-name after the execution of a PERFORM statement may be undefined.

In Format 1, the following action occurs:

- *index-name-1* is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by *index-name-2*, *identifier-2* or *integer-1*. If *identifier-2* is an index data item, see the note below.
- If *identifier-1* is an index data item, it may be set equal to the contents of either *index-name-2* or *identifier-2*, where *identifier-2* is also an index data item.
- If *identifier-1* is not an index data item, it may be set only to an occurrence number that corresponds to the value of *index-name-2*. Neither *identifier-2* nor *integer-1* can be used in this case.
- The process is repeated if specified. Any subscripting associated with *identifier-1* is evaluated immediately before the value of the respective data item is changed.

In Format 2, the contents of each *index-name-3* are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of *integer-2* or of the data item referenced by *identifier-3*. Each time the value of *identifier-3* is used as it was at the beginning of the execution of the statement.

Note Standard COBOL does not require conversion of an index value (that is, the character offset within the table to a specific occurrence of a table element) to or from the occurrence number in either case. It is an error to code the following sequence when *index-name-4* and *index-name-5* are not associated with the same table:

```
SET index-data-item TO index-name-4.
SET index-name-5 TO index-data-item.
```

Generally, RM/COBOL cannot detect such errors. It treats index data items as if they contained occurrence numbers and converts to or from index values as necessary in SET statements. Programs which depend on this conversion will not necessarily execute correctly on other implementations of standard COBOL.

Table 6-10 shows the validity of various operand combinations in the SET statement.

Table 6-10: SET Statement Operand Validity

Sending Item	Receiving Item		
	Integer Data Item	Index-Name	Index Data Item
Integer Literal	No	Valid	No
Integer Data Item	No	Valid	No
Index-Name	Valid	Valid	Valid 1
Index Data Item	No	Valid 1	Valid 2

[1] *No conversion is required in standard COBOL, but RM/COBOL converts between occurrence number for index data items and index value for index-names.*

[2] *No conversion takes place.*

In Format 3, the status of each external switch associated with the specified *mnemonic-name-1* is modified such that the truth value resultant from evaluation of a condition-name associated with that switch reflects an on status if the ON phrase is specified, or an off status if the OFF phrase is specified.

In Format 4 if the TRUE phrase is specified, the literal in the VALUE clause associated with *condition-name-1* is placed in the conditional variable according to the rules for the VALUE clause. 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.

In Format 4 if the FALSE phrase is specified, the literal in the FALSE phrase of the VALUE clause associated with *condition-name-1* is placed in the conditional variable according to the rules for the VALUE clause.

If multiple condition-names are specified in Format 4, 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.

In Format 5, the sending value represents the address of a data item. If *identifier-6* is specified, the sending value is the value of the pointer data item referred to by *identifier-6*. If ADDRESS OF *identifier-5* is specified, the sending value represents the address of the data item referred to by *identifier-5*. If NULL or NULLS is specified, the sending value is the null pointer value, which is not the address of any data item.

In Format 5, the receiving data item is either a pointer data item or the base address of a based linkage record. If *identifier-4* is specified, the receiving data item is a pointer data item into which the sending value is stored. If ADDRESS OF *data-name-1* is specified, the receiving data item is a system-defined base address pointer data item for the based linkage record. In the latter case, the object program subsequently operates as if the

based linkage record identified by *data-name-1* were located at the address represented by the sending value.

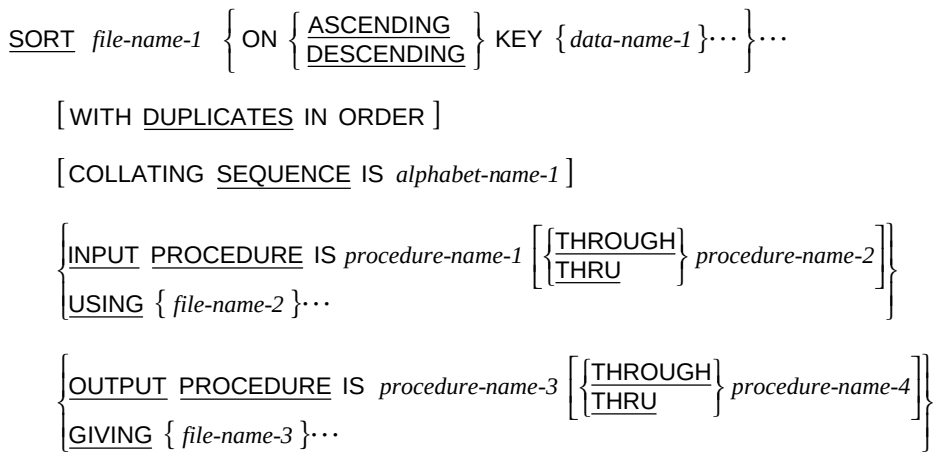
In Format 6, the UP phrase increments and the DOWN phrase decrements the offset field of a pointer receiving data item identified by *identifier-4* or the base address of a based linkage record identified by *data-name-1* by a given number of character positions. The number of character positions to increment or decrement the receiving value is given by *integer*-3, the value of the data item referred to by *identifier-7*, or the value returned by the LENGTH special register for *identifier-8*. If the receiving item initially has a null value, the Format 6 SET statement has no effect. If after the operation of the Format 6 SET statement, the offset exceeds the length field of the receiving pointer value no action is taken. However, if that resultant pointer value is used unchanged to reference a based linkage record, the run unit will be terminated with a data reference error 104. Note that, because the offset field of a pointer value is an unsigned quantity, setting it down below zero will generally result in a large positive number that exceeds the length field of the pointer value. Again, no error occurs until a later attempt is made to use the resultant pointer value.

SET Statement Examples

```
SET IX1 IX2 TO IX3, IX3 IX4 TO SUB1.

SET IX1 IX2 UP BY 1, IX3 IX4 DOWN BY 2.

SET SUMMARY-SWITCH TO OFF, DETAIL-SWITCH TO ON.

SET EOF TO TRUE, COND-1 TO FALSE.

SET P1 TO P2.

SET ADDRESS OF BL-RECORD TO P1.

SET P1 TO ADDRESS OF G1.

SET P2 TO NULL.

SET P1 UP BY LENGTH OF T1(1).

SET ADDRESS OF BL-RECORD DOWN BY COUNT-1.
```

SORT Statement

The SORT statement creates a sort file by executing an input procedure or by transferring records from another file, sorts the records in the sort file on a set of specified keys, and in the final phase of the sort operation, makes available each record from the sort file, in sorted order, to an output procedure or to an output file.



A SORT statement may appear anywhere in the Procedure Division except in the declaratives portion.

file-name-1 must be described in a sort-merge file description entry in the Data Division.

data-name-1 may be qualified.

data-name-1 must reference either a record-name associated with *file-name-1* or a data item in a record associated with *file-name-1*. If more than one record description entry is associated with *file-name-1*, the data items referenced by different specifications of *data-name-1* need not all be associated with the same record description entry.

The data item referenced by *data-name-1* must not be a group item that contains a variable-occurrence data item.

file-name-2 and *file-name-3* must be described in a file description entry in the Data Division.

The files referenced by *file-name-2* and *file-name-3* may reside on the same multiple file reel (or reels). (See the discussion of the I-O-CONTROL paragraph that begins on page 3-36.)

No pair of file-names in the same SORT statement may be specified in the same SAME SORT AREA or SAME SORT-MERGE AREA clause. (See the I-O-CONTROL paragraph.)

The words THRU and THROUGH are synonymous.

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 shorter than the shortest record nor longer than the longest 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 longer than the fixed record size specified for the file referenced by *file-name-1*.

If the GIVING phrase is specified and the file referenced by *file-name-3* contains variable-length records, the size of the records contained in the file referenced by *file-name-1* must not be shorter than the shortest record nor longer than the longest record described for *file-name-3*. If the file referenced by *file-name-3* contains fixed-length records, the size of the records contained in the file referenced by *file-name-1* must not be longer than the fixed record size specified for the file referenced by *file-name-3*.

General Rules

The general rules that apply to the SORT statement are as follows:

1. The SORT statement releases all the records in the file referenced by *file-name-2* or released by an input procedure to the file referenced by *file-name-1*, and returns them to an output procedure, or to the file referenced by *file-name-3*, in an order determined by the ASCENDING and DESCENDING phrases and the values of the data items referenced by the specifications of *data-name-1*.

2. The words ASCENDING and DESCENDING apply to each subsequent occurrence of *data-name-1* until another word ASCENDING or DESCENDING is encountered.

3. The data items referenced by the specifications of *data-name-1* are the key data items that determine the order in which records are returned from the file referenced by *file-name-1*. The order of significance of the keys is the order in which they are specified in the SORT statement, without regard to their association with ASCENDING or DESCENDING phrases. The first (or only) key data item is the most significant. Further key data items, if any, are of progressively lesser significance.

4. To determine the relative order in which two records are returned from the file referenced by *file-name-1*, the contents of corresponding key data items are compared according to the rules for comparison of operands in a relation condition, starting with the most significant key data item.

    a. If the contents of the corresponding key data items are not equal and the key is associated with the ASCENDING phrase, the record containing the key data item with the lower value is returned first.

    b. If the contents of the corresponding key data items are not equal and the key is associated with the DESCENDING phrase, the record containing the key data item with the higher value is returned first.

    c. If the contents of the corresponding key data items are equal, the determination is made on the contents of the next most significant key data item.

5. If the DUPLICATES phrase is specified and the contents of all the key data items in one record are equal to the contents of the corresponding key data items in one or more other records, the order of the return of such duplicate-key records is:

    a. When an input procedure is not specified, the order of the associated input files is specified in the SORT statement. Within a given input file the order is that in which the records are accessed from that file.

    b. When an input procedure is specified, the order in which these records are released by that input procedure.

6. If the DUPLICATES phrase is not specified, the order in which duplicate-key records are returned is not predictable.

7. The collating sequence that applies to the comparison of nonnumeric key data items is determined at the beginning of the execution of the SORT statement in the following order of precedence:

    a. The collating sequence established by the COLLATING SEQUENCE phrase, if specified, in the SORT statement.

    b. The collating sequence established as the program collating sequence.

8. 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-3* takes place during this phase.

c. The records of the file referenced by *file-name-1* are made available in sorted order. The sorted records are either written to the file referenced by *file-name-3* or, by the execution of a RETURN statement, are made available for processing by the output procedure. When this phase commences, the file referenced by *file-name-3* must not be in the open mode. When this phase terminates, the file referenced by *file-name-3* is not in the open mode.

9. The input procedure may consist of any procedure needed to select, modify or copy the records that are made available one at a time by the RELEASE statement to the file referenced by *file-name-1*. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT without the optional PROGRAM phrase, GO TO and PERFORM statements in the range of the input procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the input procedure. The range of the input procedure must not cause the execution of any MERGE, RETURN or SORT statement.

10. If an input procedure is specified, control is passed to the input procedure before the file referenced by *file-name-1* is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last statement in the input procedure and when control passes the last statement in the input procedure, the records that have been released to the file referenced by *file-name-1* are sorted.

11. If the USING phrase is specified, all the records in the file (or files) referenced by *file-name-2* are transferred to the file referenced by *file-name-1*. For each of the files referenced by *file-name-2* the execution of the SORT statement causes the following actions to be taken:

    a. The processing of the file is initiated. 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 the AT END phrases had been executed. If the file referenced by *file-name-1* contains fixed-length records, any record in the file referenced by *file-name-2* containing fewer character positions than that specified for *file-name-1* is space-filled on the right beginning with the first character position after the last character in the record when that record is released to the file referenced by *file-name-1*.

c. For a relative file, the contents of the relative key data item are undefined after the execution of the SORT statement if *file-name-2* is not referenced in the GIVING phrase.

d. 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 sequenced by the SORT statement.

These implicit functions are performed such that any associated USE procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the file referenced by *file-name-2* or accessing the record area associated with *file-name-2*.

12. The output procedure may consist of any procedure needed to select, modify or copy the records that are made available one at a time by the RETURN statement in sorted order from the file referenced by *file-name-1*. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT without the optional PROGRAM phrase, GO TO and PERFORM statements in the range of the output procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure. The range of the output procedure must not cause the execution of any MERGE, RELEASE, or SORT statement.

13. If an output procedure is specified, control passes to it after the file referenced by *file-name-1* has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last statement in the output procedure and when control passes the last statement in the output procedure, the return mechanism provides for termination of the sort and then passes control to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

14. If the GIVING phrase is specified, all the sorted records are written on the file referenced by *file-name-3* as the implied output procedure for the SORT statement. At the start of execution of the SORT statement, the file referenced by *file-name-3* must not be in the open mode. For each of the files referenced by *file-name-3*, the execution of the SORT statement causes the following actions to be taken:

    a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed. The initiation occurs after the execution of the input procedure, if there is one.

b. The sorted logical records are returned and written onto the file. Each record is written as if a WRITE statement without any optional phrases had been executed.

c. 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, and so forth. After execution of the SORT statement, the contents of the relative key data item indicate the last record returned to the file.

d. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

These implicit functions are performed such that any associated USE procedures are executed. However, the execution of such a USE procedure must not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, *file-name-3*. On the first attempt to write beyond the externally defined boundaries of the file, any USE procedure specified for the file is executed. If control is returned from that USE procedure or if no such USE procedure is specified, the processing of the file is terminated as described above.

15. 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, any input procedures or output procedures referenced by that SORT statement must appear:

        (1) Totally within nonindependent segments, or

        (2) Wholly contained in a single independent segment.

    b. If a SORT statement appears in an independent segment, any input procedures or output procedures referenced by that SORT statement must be contained:

        (1) Totally within nonindependent segments, or

        (2) Wholly within the same independent segment as that SORT statement.

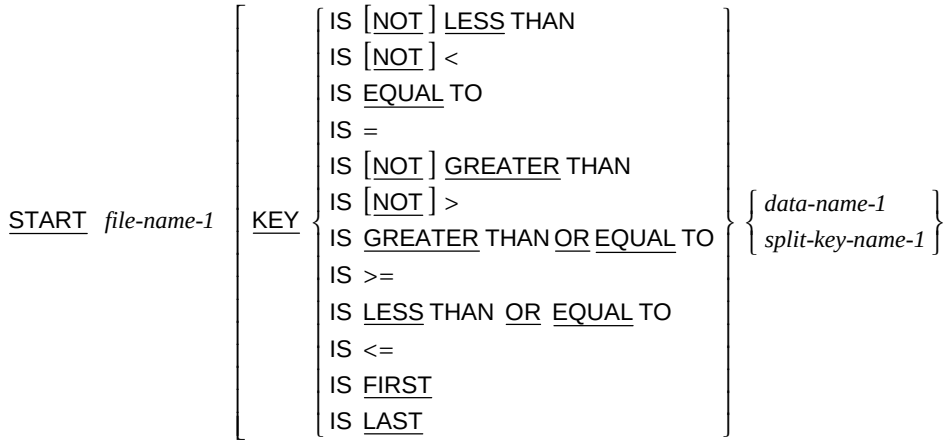
SORT Statement Example

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  SORT01.
*
*  Examples for RM/COBOL Language Reference Manaual.
*  SORT statement.
*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT SORT-FILE ASSIGN TO SORT-WORK.

 DATA DIVISION.
 FILE SECTION.
 SD SORT-FILE.
 01 SORT-RECORD.
    02 SORT-KEY-1         PIC X(05).
    02 SORT-DATA-1        PIC X(20).
    02 SORT-KEY-2         PIC 9(05) BINARY.
 WORKING-STORAGE SECTION.
 01 EOF-FLAG              PIC X.
    88 EOF                VALUE "T" FALSE "F".



 PROCEDURE DIVISION.
 MAIN1.
     SORT SORT-FILE
       ON ASCENDING KEY SORT-KEY-1
       ON DESCENDING KEY SORT-KEY-2
       WITH DUPLICATES IN ORDER
       INPUT PROCEDURE IS GET-RECORDS
       OUTPUT PROCEDURE IS PUT-RECORDS.
     STOP RUN.

 GET-RECORDS.
     PERFORM WITH TEST AFTER UNTIL EOF
       CALL "READ-RECORD" USING SORT-RECORD, EOF-FLAG
       IF NOT EOF
         RELEASE SORT-RECORD
       END-IF
     END-PERFORM.
```

```
PUT-RECORDS.
    SET EOF TO FALSE.
    PERFORM UNTIL EOF
      RETURN SORT-FILE RECORD
      AT END SET EOF TO TRUE
      NOT AT END
        CALL "WRITE-RECORD" USING SORT-RECORD
      END-RETURN
    END-PERFORM.

END PROGRAM SORT01.
```

START Statement (Relative and Indexed I-O)

The START statement provides a method of setting the file position indicator for a relative or indexed file and provides a means of determining whether a record exists.



Note The required relational characters >, < and = are not underlined to avoid confusion with other symbols.

file-name-1 must be the name of a relative or indexed file.

file-name-1 must be the name of a file with sequential or dynamic access.

data-name-1 or *split-key-name-1* may be qualified.

split-key-name-1 and the SIZE phrase may be specified only for indexed files.

identifier-1 must refer to an elementary integer data item.

The INVALID KEY phrase must be specified if no applicable USE procedure is specified for *file-name-1*.

If *file-name-1* is the name of a relative file then *data-name-1*, if specified, must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.

If *file-name-1* is the name of an indexed file then *data-name-1*, if specified, may reference any one of the data items specified as the record keys associated with *file-name-1* or it may reference any data item of category alphanumeric whose leftmost character position corresponds to the leftmost character position of a record key data item and whose length is not greater than the length of that record key.

If *file-name-1* is the name of an indexed file, *split-key-name-1*, if specified, may refer to any one of the split keys specified as the record keys associated with *file-name-1*.

file-name-1 must be open in the INPUT or I-O mode at the time the START statement is executed.

If the KEY phrase is not specified, the relational operator IS EQUAL TO is implied and, for an indexed file, the key of reference is the prime record key of the file.

The type of comparison specified by the relational operator in the KEY phrase of a START statement occurs between a key associated with a record in the file to which *file-name-1* refers and a data item.

- If *file-name-1* refers to a relative file, the data item used in the comparison is the relative key associated with *file-name-1*. All numeric comparison rules apply.

- If *file-name-1* refers to an indexed file, the data item used in the comparison is either the prime record key associated with *file-name-1* or, if the KEY phrase is specified, the data item or split key to which the KEY phrase refers. The comparison is made on the ascending key of reference according to the collating sequence of the file. If the operands of the comparison are of unequal size, comparison proceeds as though the longer one were truncated on the right such that its length is equal to that of the shorter. The size of the comparison is further modified by the SIZE phrase, if specified. All other nonnumeric comparison rules apply, except that the presence of the PROGRAM COLLATING SEQUENCE clause has no effect on the comparison.

When FIRST or LAST are specified in the KEY phrase instead of a relational operator, no comparison takes place and the value of the relative key data item for a relative file or the value of the key of reference data item for an indexed file is not used in setting the file position indicator.

For a relative file the file position indicator is modified as follows:

- If the relational operator specifies that the key must be “equal to”, “greater than” or “greater than or equal to” the data item, then the file position indicator is set to the lowest relative record number of a record currently existing in the file whose key satisfies the comparison.
- 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 is set to the highest relative record number of a record currently existing in the file whose key satisfies the comparison.
- If FIRST is specified, the file position indicator is set to the lowest relative record number of a record currently existing in the file.
- If LAST is specified, the file position indicator is set to the highest relative record number of a record currently existing in the file.

For an indexed file, the file position indicator is modified as follows:

- If the relational operator specifies that the key must be “equal to”, “greater than” or “greater than or equal to” the data item, then the file position indicator is set to the value of the key of reference of the first logical record currently existing in the file whose key satisfies the comparison.
- If the relational operator specifies that the key must be “less than” the data item, then the file position indicator is set to the value of the key of reference of the last logical record currently existing in the file whose key satisfies the comparison.
- If the relational operator specifies that the key must be “less than or equal to” the data item, then the file position indicator is set to the value of the key of reference of the first record whose key equals the data item. If no record with the specified key value currently exists in the file, then the file position indicator is set to the value of the key of reference of the last logical record currently existing in the file whose key satisfies the comparison.
- If FIRST is specified, the file position indicator is set to the lowest value of the key of reference of a record existing in the file according to the collating sequence of the file.
- If LAST is specified, the file position indicator is set to the highest value of the key of reference of a record existing in the file according to the collating sequence of the file.

If there are no records currently existing in the file or if the comparison is not satisfied by any record currently existing in the file, an invalid key condition exists. The invalid key condition also exists if *file-name-1* refers to an optional input file that is not present. When the invalid key condition exists, the execution of the START statement is unsuccessful, the file position indicator is set to indicate that no valid next record has been established, and, for indexed files, the key of reference becomes undefined.

The execution of the START statement causes the value of the file status data item associated with *file-name-1*, if there is one, to be updated. It does not alter either the contents of the record area or the contents of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with *file-name-1*.

For indexed files, a key of reference is established as follows:

- If the KEY phrase is not specified, the prime record key for the file becomes the key of reference.
- If the KEY phrase is specified and *data-name-1* or *split-key-name-1* is one of the record keys of the file, that record key becomes the key of reference.
- If the KEY phrase is specified and *data-name-1* is not one of the record keys of the file, the record key whose leftmost character position coincides with the leftmost character position of the data item referenced by *data-name-1* becomes the key of reference.

For indexed files, the key of reference is used to select the data item that participates in the key comparison described above, and it is used for subsequent sequential (Format 1) READ statements.

In single record locking modes, any record lock held by the run unit for *file-name-1* is released upon execution of the START statement. The START statement does not obtain a record lock and does not indicate the lock status of the record that satisfies the comparison.

In multiple record locking modes, any record locks held by the run unit for *file-name-1* are not released upon execution of the START statement.

SIZE Phrase

```
WITH SIZE { identifier-1 }
          { integer-1    }
```

The SIZE phrase modifies the length of the data item used in the comparison to a key associated with a record in the indexed file to which *file-name-1* refers. Since there is no comparison to a data item for the FIRST and LAST options, the SIZE phrase has no effect when specified with those options. The SIZE phrase is not allowed if *file-name-1* refers to a relative file.

When the SIZE phrase is omitted, the size of the data item specified in the KEY phrase, or the size of the prime record key for *file-name-1* when the KEY phrase is omitted, is used as the size in the comparison described above.

When the SIZE phrase is present, *integer-1* or the value of the data item to which *identifier-1* refers is used as the size in the comparison described above.

integer-1 or the value of the data item to which *identifier-1* refers must be greater than or equal to one and less than or equal to the length of the record key specified by the KEY phrase, if present, or the length of the prime record key for *file-name-1*, if the KEY phrase is omitted.

Note Specification of the SIZE phrase overrides the size of a data item specified by *data-name-1* when that data item is not a record key of the file.

INVALID KEY and NOT INVALID KEY Phrases

```
INVALID KEY imperative-statement-1

NOT INVALID KEY imperative-statement-2
```

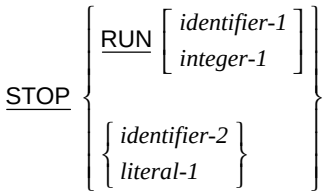
The causes of the invalid key condition for the START statement are indicated in the preceding text. Transfer of control following the execution of the START operation depends on the presence or absence of the INVALID KEY and NOT INVALID KEY phrases. See the discussions of relative organization input-output and indexed organization input-output on pages 5-50 and 5-59 for additional information regarding the invalid key condition and the effect of the INVALID KEY phrases.

START Statement (Relative and Indexed I-O) Examples

```
MOVE 10 TO INVENTORY-KEY.
START INVENTORY-FILE; INVALID KEY
  DISPLAY "Key 10 not present in inventory file."
NOT INVALID KEY
  DISPLAY "Key 10 present in inventory file."
END-START.


START STATUS-FILE KEY IS LAST SF-KEY.


MOVE DB-START-KEY TO DB-KEY.
START DATA-BASE KEY >= DB-KEY SIZE 10
INVALID KEY PERFORM DB-INVALID-KEY-HANDLER
NOT INVALID KEY PERFORM DB-SUCCESS-HANDLER
END-START.
```

STOP Statement

The STOP statement causes a permanent or temporary suspension of the execution of the object program.



The implicit or explicit usage of both *identifier-1* and *identifier-2* must be DISPLAY.

literal-1 may be numeric or nonnumeric or may be any figurative constant.

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.

If the RUN phrase is used:

- The execution of the entire run unit is terminated.
- *integer-1* or the value of the data item referenced by *identifier-1* may be zero.
- When *identifier-1* is used in the RUN phrase, it must be the name of an elementary integer data item.
- The value of the data item referenced by *identifier-1* or the value of *integer-1* is used to set the RETURN-CODE special register. When the run unit is terminated by a STOP RUN or GOBACK statement, the value in the RETURN-CODE special register is made available to the operating system. See the *RM/COBOL Use's Guide* for details on using that value.

There is an implicit interaction between the STOP RUN statement and the RETURN-CODE special register. See the discussion of the RETURN-CODE special register on page 1-13.

If STOP *identifier-2* or *literal-1* is specified, the value of the operand is displayed at the terminal associated with this run unit and execution of the run unit is suspended until the message is acknowledged. After the message is acknowledged, execution continues with the next executable statement.

STOP Statement Examples

```
STOP RUN.

STOP RUN 1.

STOP RUN STATUS-CODE.

STOP "End of Procedure.".
```

STRING Statement

The STRING statement concatenates the partial or complete contents of one or more data items into a single data item.

INTO *identifier-3*

[WITH POINTER *identifier-4*]

[ON OVERFLOW *imperative-statement-1*]

[NOT ON OVERFLOW *imperative-statement-2*]

[END-STRING]

literal-1 and *literal-2* may be any figurative constant except those that begin with the word ALL. When figurative constants are used in a STRING statement, they behave as single character nonnumeric literals.

All literals must be nonnumeric literals, and the explicit or implicit usage of each identifier, except *identifier-4*, must be DISPLAY.

identifier-3 must not be reference modified; it must not represent an edited data item; and it must not be described with the JUSTIFIED clause.

identifier-4 must represent an elementary numeric integer data item of sufficient size to contain a value equal to the size plus 1 of the area referenced by *identifier-3*. The symbol P may not be used in the PICTURE character-string of *identifier-4*.

When *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.

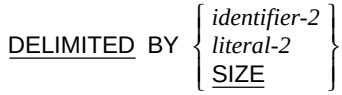
identifier-1 or *literal-1* represents the sending item. *identifier-3* in the INTO phrase represents the receiving item.

When the STRING statement is executed, characters from *literal-1* or from the contents of the data item referenced by *identifier-1* are transferred to the data item referenced by *identifier-3* in accordance with the rules for alphanumeric to alphanumeric moves, except that no space filling is provided.

When characters are transferred to the data item referenced by *identifier-3*, the moves behave as though the characters were moved one at a time from the source into the character position of the data item referenced by *identifier-3* designated by the value associated with *identifier-4*, and then *identifier-4* was 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 according to the rules set forth in the POINTER phrase description.

At the end of the execution of the STRING statement, only the portion of the data item referenced by *identifier-3* that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by *identifier-3* contain data that was present before this execution of the STRING statement.

DELIMITED Phrase



If the DELIMITED phrase is specified without the SIZE phrase, the contents of the data item referenced by *identifier-1* or the value of *literal-1* are transferred to the receiving data item in the sequence specified in the STRING statement beginning with the leftmost character and continuing from left to right until the end of the sending data item is reached, or the end of the receiving data item is reached, or until the character (or characters) specified by *literal-2*, or by the content of the data item referenced by *identifier-2* is encountered. The character (or characters) specified by *literal-2* or by the data item referenced by *identifier-2* is not transferred.

If the DELIMITED phrase is specified with the SIZE phrase, the entire contents of *literal-1* or the contents of the data item referenced by *identifier-1* are transferred, in the sequence specified in the STRING statement, to the data item referenced by *identifier-3* until all data has been transferred or the end of the data item referenced by *identifier-3* has been reached.

POINTER Phrase

WITH POINTER *identifier-4*

If the POINTER phrase is specified, the data item referenced by *identifier-4* must have a positive value at the time execution of the STRING statement begins.

If the POINTER phrase is not specified, the effect is as if the user had specified *identifier-4* referencing a data item with an initial value of 1.

OVERFLOW and NOT OVERFLOW Phrases

ON OVERFLOW *imperative-statement-1*

NOT ON OVERFLOW *imperative-statement-2*

Before each move of a character from the current sending item to the receiving item, if the value associated with the data item referenced by *identifier-4* is either less than one or exceeds the number of character positions in the receiving item, an overflow condition exists.

If an overflow condition arises, no (further) data is transferred from the sending item to the receiving item, the NOT ON OVERFLOW phrase, if present, is ignored, and control is transferred either to the end of the STRING statement, or, if the ON OVERFLOW phrase is present, to *imperative-statement-1*. In the latter case, 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 encountered, 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.

If the STRING statement executes without an overflow condition arising, the ON OVERFLOW phrase, if present, is ignored and control is transferred either to the end of the STRING statement, or, if the NOT ON OVERFLOW phrase is present, to *imperative-statement-2*. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement that causes explicit transfer of control is encountered, 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.

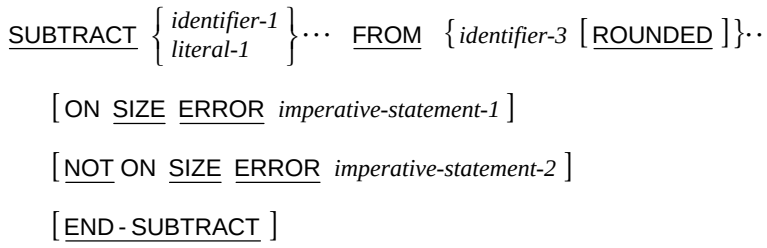
STRING Statement Examples

```
STRING FIELD-1 DELIMITED BY SPACES
    ";" DELIMITED BY SIZE
    FIELD-2 DELIMITED BY "."
    ";" DELIMITED BY SIZE
  INTO FIELD-GROUP
ON OVERFLOW
  DISPLAY "Overflow error."
  STOP RUN
END-STRING.


STRING MONTH-VALUE DELIMITED BY SPACES
    SPACE DAY-VALUE "," YEAR-VALUE
    DELIMITED BY SIZE
  INTO TITLE-RECORD
  WITH POINTER COLUMN-CURSOR.
```

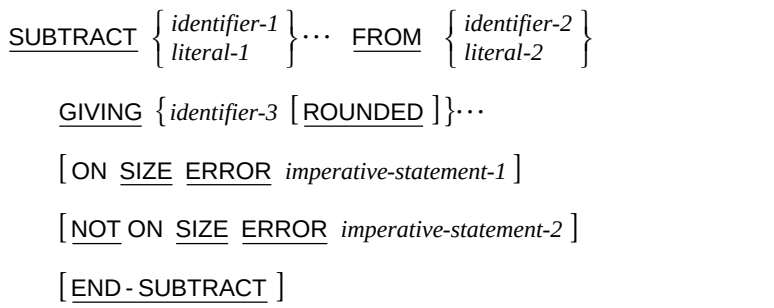
SUBTRACT Statement

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from a numeric data item and store the result.

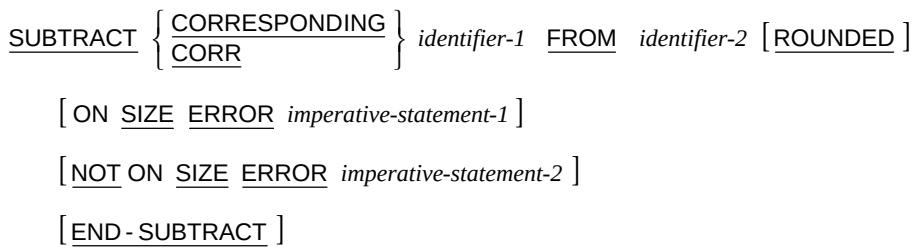
Format 1: Subtract…From



Format 2: Subtract…Giving



Format 3: Subtract Corresponding



In Format 1, all literals or identifiers preceding the word FROM are added together and the sum is stored in a temporary data item. The value of this temporary data item is subtracted from the value of the data item specified by *identifier-3*, storing the result into the data item specified by *identifier-3*, and repeating this process for each successive occurrence of *identifier-3* in the left-to-right order in which *identifier-3* is specified.

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 *identifier-3*.

If Format 3 is used, data items in *identifier-1* are subtracted from and stored into corresponding data items in *identifier-2*.

Each identifier must refer to a numeric elementary item except that:

- In Format 2, the identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.
- In Format 3, the identifiers must refer to group items.

Each literal must be a numeric literal.

Additional rules and explanations regarding features of the SUBTRACT statement that are common to other arithmetic statements can be found in the discussion of common rules beginning on page 5-16. See in particular the discussions of the ROUNDED phrase, the size error condition, overlapping operands, modes of operation, composite size, and incompatible data.

CORRESPONDING Phrase

{ CORRESPONDING / CORR } *identifier-1* FROM *identifier-2* [ROUNDED]

If the CORRESPONDING phrase is used, selected items within *identifier-1* are subtracted from, and the result stored in, the corresponding items in *identifier-2*.

For the SUBTRACT statement with the CORRESPONDING phrase:

- The description of *identifier-1* and *identifier-2* must not contain level-number 66, 77, 78, or 88, or the USAGE IS INDEX clause.
- Neither *identifier-1* nor *identifier-2* may be reference modified.
- *identifier-1* or *identifier-2* may be described with the OCCURS or REDEFINES clauses or be subordinate to data items described with the OCCURS or REDEFINES clauses. If *identifier-1* or *identifier-2* is a table element, then the required

subscripting must be specified as part of *identifier-1* or *identifier-2*. The specified subscripting will be applied to the selected subordinate corresponding data items, respectively, for *identifier-1* and *identifier-2*.

The rules that govern the selection of eligible subordinate data item pairs are as follows:

1. The data items are not designated by the keyword FILLER and have the same *data-name-1* and the same qualifiers up to but not including the original group items, *identifier-1* and *identifier-2*.

2. Both of the data items are elementary numeric data items.

3. A data item that is subordinate to *identifier-1* or *identifier-2* 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.

4. The name of each data item that satisfies the above conditions must be unique after application of the implied qualifiers.

If any of the individual operations produces a size error condition, *imperative-statement-1* in the ON SIZE ERROR phrase is not executed until all of the individual subtractions are completed.

CORR is an abbreviation for CORRESPONDING.

SUBTRACT Statement Examples

```
SUBTRACT TAXES FROM INCOME.


SUBTRACT 1 FROM TALLY-COUNTER GIVING TALLY-1.


SUBTRACT 2.68, INTEREST, PENALTY
  FROM PRINCIPAL ROUNDED
ON SIZE ERROR GO TO ERROR-HANDLER.


SUBTRACT CORR DAILY-SALES FROM INVENTORY-ON-HAND.
```

UNLOCK Statement

The UNLOCK statement releases all record locks held by the run unit for a shared input-output file.

```
UNLOCK  file-name-1  [ RECORD  ]
                     [ RECORDS ]
```

The file to which *file-name-1* refers must be in an open mode at the time the UNLOCK statement is executed.

In all record locking modes any record locks held by the run unit for *file-name-1* are released upon execution of the UNLOCK statement.

If no record in the file is locked, execution of the UNLOCK statement is successful and no action is taken except for updating the file status data item.

The file position indicator is not affected by the execution of the UNLOCK statement. The file status data item associated with the file, if one exists, is updated.

The UNLOCK statement may not be used to unlock records locked by other run units.

See “Record Locking” on page 5-69 for additional information on record locking and unlocking.

UNLOCK Statement Examples

```
UNLOCK DATA-BASE RECORDS.

UNLOCK INVENTORY-FILE.
```

UNSTRING Statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed in multiple receiving fields.

```
UNSTRING identifier-1

  [ DELIMITED BY [ ALL ] { identifier-2 } [ OR [ ALL ] { identifier-3 } ]... ]
                         { literal-1    }              { literal-2    }

  INTO { identifier-4 [ DELIMITER IN identifier-5 ] [ COUNT IN identifier-6 ] }...

  [ WITH POINTER identifier-7 ]

  [ TALLYING IN identifier-8 ]

  [ ON OVERFLOW imperative-statement-1 ]

  [ NOT ON OVERFLOW imperative-statement-2 ]

  [ END-UNSTRING ]
```

literal-1 and *literal-2* must be nonnumeric literals and may be any figurative constant except those that begin with the word ALL.

identifier-1, *identifier-2*, *identifier-3* and *identifier-5* must reference data items described implicitly or explicitly as category alphanumeric.

identifier-1 must not be reference modified.

identifier-4 may be described as alphabetic, 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.

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).

The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

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 referenced by *identifier-2* are treated as if they were only one occurrence, and one occurrence of *literal-1* or the data item referenced by *identifier-2* is moved to the receiving data item.

When the ALL phrase is not specified and any examination encounters two contiguous delimiters, the current receiving area is space filled if it is described as alphabetic or alphanumeric, or zero filled if it is described as numeric.

literal-1 or the contents of the data item referenced by *identifier-2* can contain any character in the character set of the computer.

Each *literal-1* or the data item referenced by *identifier-2* represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item, and in the order given, to be recognized as a delimiter.

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 (or characters) in the sending field is considered to be a single delimiter. No character (or characters) 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.

When the UNSTRING statement is initiated, the current receiving area is the data item referenced by *identifier-4*. Data is transferred from the data item referenced by *identifier-1* to the data item referenced by *identifier-4* according to the following rules:

1. If the POINTER phrase is specified, the string of characters referenced by *identifier-1* is examined beginning with the relative character position indicated by the contents of the data item referenced by *identifier-7*. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.

2. 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 referenced by *identifier-2* is encountered. 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 referenced by *identifier-1* is encountered before the delimiting condition is met, the examination terminates with the last character examined.

3. 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; see page 6-113.

4. If the DELIMITER IN phrase is specified, the delimiting character (or characters) are treated as an elementary alphanumeric data item, and are moved into the data item referenced by *identifier-5* according to the rules for the MOVE statement. If the delimiting condition is the end of the data item referenced by *identifier-1*, the data item referenced by *identifier-5* is space filled.

5. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter characters, if there are any) is moved into the area referenced by *identifier-6* according to the rules for an elementary move.

6. 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.

7. After data is transferred to the data item referenced by *identifier-4* in the INTO phrase, the current receiving area is the data item referenced by the next recurrence of *identifier-4*. Steps 2 through 6 above are then repeated until all the characters are exhausted in the data item referenced by *identifier-1*, or until there are no more receiving areas.

The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

The contents of the data item referenced by *identifier-7* are incremented by one for each character examined in the data item referenced by *identifier-1*. When the execution of an UNSTRING statement with a POINTER phrase is completed, the contents of the data item referenced by *identifier-7* contain a value equal to the initial value plus the number of characters examined in the data item referenced by *identifier-1*.

When the execution of an UNSTRING statement with a TALLYING phrase is completed, the data item referenced by *identifier-8* contains a value equal to its initial value plus the number of data receiving items acted upon.

Either of the following situations causes an overflow condition:

- An UNSTRING statement is initiated, and the value in the data item referenced by *identifier-7* is less than 1 or greater than the size of the data item referenced by *identifier-1*.
- If, during the execution of an UNSTRING statement, all data receiving areas have been acted upon, and the data item referenced by *identifier-1* contains characters that have not been examined.

If an overflow condition arises, the UNSTRING operation is terminated, the NOT ON OVERFLOW phrase, if present, is ignored, and control is transferred either to the end of the UNSTRING statement, or, if the ON OVERFLOW phrase is present, to *imperative-statement-1*. In the latter case, 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 encountered, 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.

If the UNSTRING operation completes without an overflow condition arising, the ON OVERFLOW phrase, if present, is ignored and control is transferred either to the end of the UNSTRING statement, or, if the NOT ON OVERFLOW phrase is present, to *imperative-statement-2*. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement that causes explicit transfer of control is encountered, 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.

UNSTRING Statement Example

```
MOVE ZERO TO FIELD-COUNT.
UNSTRING PARAMETER-1 DELIMITED BY ";" OR "."
  INTO FIELD-1 DELIMITER IN DELIM-1
       FIELD-2 DELIMITER IN DELIM-2
       FIELD-3 DELIMITER IN DELIM-3
  TALLYING IN FIELD-COUNT
ON OVERFLOW
  DISPLAY "Too many fields in parameter."
  STOP RUN
END-UNSTRING.
```

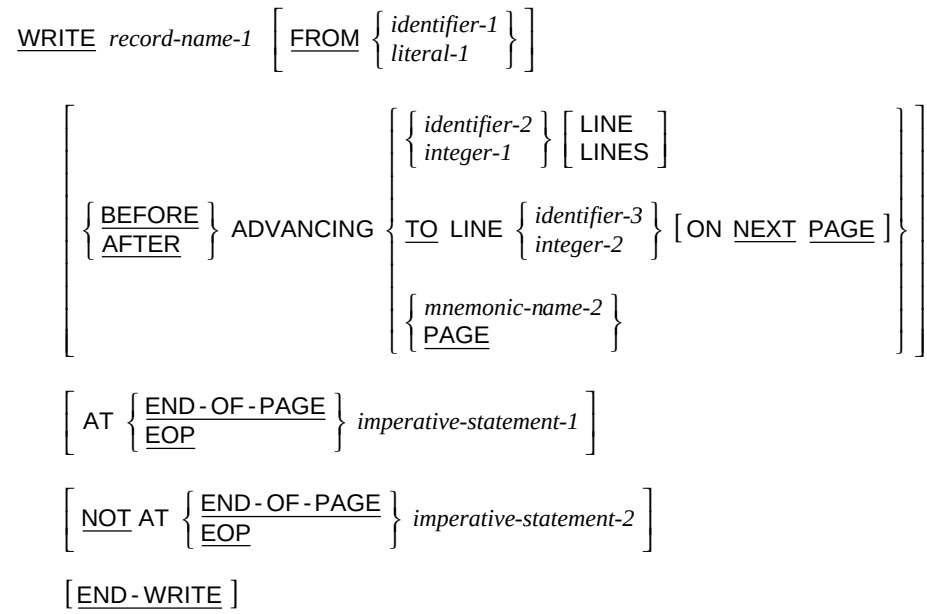
USE Statement

The discussion of the USE statement begins on page 5-13.

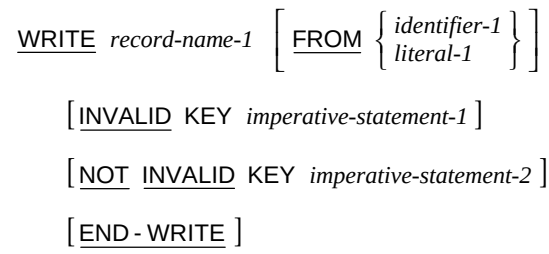
WRITE Statement

The WRITE statement releases a logical record for an output or input-output file. For a sequential file, it can also be used for vertical positioning of lines within a logical page.

Format 1: Write Sequential File



Format 2: Write Relative and Indexed File



In a Format 1 WRITE statement, *record-name-1* must refer to a record associated with a sequential organization file.

In a Format 2 WRITE statement, *record-name-1* must refer to a record associated with a relative or indexed organization file.

record-name-1 and *identifier-1* must not reference the same storage area.

record-name-1 is the name of a logical record in the File Section of the Data Division and may be qualified.

When *identifier-2* is used in the ADVANCING phrase, it must be the name of an elementary integer data item.

integer-1 or the value of the data item referenced by *identifier-2* may be zero.

identifier-3 must reference an unsigned integer data item.

In a Format 2 WRITE statement, the INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

If the access mode is sequential, the associated file must be open in the OUTPUT or EXTEND mode at the time of the execution of this statement.

If the access mode is random or dynamic, the associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement.

The file position indicator is unaffected by the execution of a WRITE statement.

The execution of the WRITE statement causes the value of the file status data item, if any, associated with the file to be updated.

The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

The execution of the WRITE statement releases a logical record to the operating system. The contents of the record area are not changed.

When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists. The following action takes place:

- The value of the file status data item, if any, of the associated file is set to a value indicating a boundary violation.
- If a USE declarative is explicitly or implicitly specified for the file, that declarative procedure is executed.
- If a USE declarative is not explicitly or implicitly specified for the file, an error message is displayed and the run unit is terminated.

When a relative file is opened in the output mode, records may be placed into the file by one of the following:

- If the access mode is sequential, the WRITE statement causes a record to be released to the associated file. The first record has a relative record number of 1, and subsequent records have relative record numbers 2, 3, 4, ... If a 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 runtime system during execution of the WRITE statement.
- 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 to be associated with the record in the record area. That record is then released to the associated file by execution of the WRITE statement.

When a relative file is opened in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. 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 to the associated file.

For an indexed file, 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. Records may be placed into the file by one of the following:

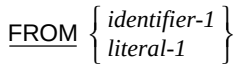
- If the access mode is sequential, records must be released in strictly ascending order of prime record key values according to the collating sequence of the file, except that, if the DUPLICATES phrase is specified in the RECORD KEY clause, records with duplicate prime record key values may be released. If the access mode is random or dynamic, records may be released to the system in any program-specified order.
- When the DUPLICATES phrase is specified for a record key of an indexed file, the value of the record key may be nonunique. In this case, the indexed file provides

storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which they are released to the runtime system.

In single record locking modes any record lock held by the run unit for the file associated with *record-name-1* is released upon execution of the WRITE statement.

In multiple record locking modes any record locks held by the run unit for *file-name-1* are not released upon execution of the WRITE statement.

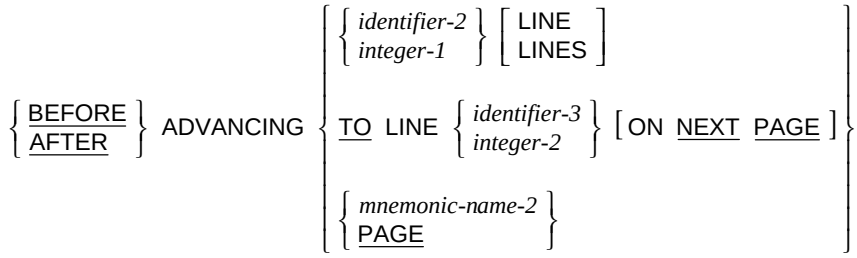
FROM Phrase



The result of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of a move from *identifier-1* or *literal-1* to *record-name-1* followed by 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.

ADVANCING Phrase



The ADVANCING phrase allows control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing occurs as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

- 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*, which must be positive or zero.
- If *integer-1* is specified, the representation of the printed page is advanced the number of lines equal to the value of *integer-1*.

- When *mnemonic-name-2* is used, the name must be identified with a feature-name that is a channel-name in the SPECIAL-NAMES paragraph of the Environment Division. The representation of the printed page is advanced to the next occurrence of the channel indicator for the channel number associated with *mnemonic-name-2*. If the print device does not support channel skipping, advancing defaults to ADVANCING 1 LINE. The mnemonic-name phrase may not be used when writing a record to a file whose file description entry contains a LINAGE clause.

- If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced.

- If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced.

- If PAGE is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page.

- If the TO LINE phrase without the NEXT PAGE phrase is specified, the representation of the printed page is positioned to the line within the current page body corresponding to *integer-2* or the value of the data item referenced by *identifier-3*.

- If the TO LINE phrase with the NEXT PAGE phrase is specified, the representation of the printed page is positioned to the line within the next logical page body corresponding to *integer-2* or the value of the data item referenced by *identifier-3*.

- If PAGE is specified and the LINAGE clause is specified in the associated file description entry, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page. The repositioning is to the first line that can be written on the next logical page as specified in the LINAGE clause.

- If PAGE is specified and the LINAGE clause is not specified in the associated file description entry, the record is repositioned to the next physical page. If physical page has no meaning in conjunction with a specific device, advancing occurs as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

END-OF-PAGE and NOT END-OF-PAGE Phrases

AT { END-OF-PAGE / EOP } *imperative-statement-1*

NOT AT { END-OF-PAGE / EOP } *imperative-statement-2*

If the END-OF-PAGE phrase, the NOT END-OF-PAGE phrase or the ADVANCING TO LINE phrase is specified, the LINAGE clause must be specified in the file description entry for the associated file.

The words END-OF-PAGE and EOP are synonymous.

An end-of-page condition occurs when the execution of a WRITE statement with the END-OF-PAGE phrase causes printing or spacing within the footing area of a page body. This occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by *integer-8* or the data item referenced by *data-name-5* of the LINAGE clause. In this case, the WRITE statement is executed and then *imperative-statement-1* in the END-OF-PAGE phrase is executed. A NOT END-OF-PAGE phrase, if present, is ignored.

An automatic page overflow condition occurs when the execution of a WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body. An automatic page overflow condition does not occur as a result of the execution of a WRITE statement containing a NEXT PAGE phrase.

An automatic page overflow condition occurs when the execution of a WRITE statement causes the LINAGE-COUNTER to exceed the value specified by *integer-7* or the data item referenced by *data-name-4* 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. *imperative-statement-1* in the END-OF-PAGE phrase, if specified, is executed after the record is written and the device has been repositioned.

A page overflow condition occurs when the execution of a WRITE statement causes the LINAGE-COUNTER to simultaneously exceed the value of both *integer-8* or the data item referenced by *data-name-5* of the LINAGE clause and *integer-7* or the data item referenced by *data-name-4* of the LINAGE clause.

If the execution of a WRITE statement with the TO LINE phrase would cause the record to be presented on a line outside the current page body if the NEXT PAGE phrase is not specified, or outside the next page body if the NEXT PAGE phrase is specified, the execution of the WRITE statement is unsuccessful. Furthermore, if the execution of the WRITE statement with *identifier-2* or *integer-1* LINES phrase would cause the

LINAGE-COUNTER associated with *record-name-1* to have a negative or zero value, the execution of the WRITE statement is unsuccessful. If the execution of the WRITE statement is unsuccessful for one of these reasons, an exception condition exists, the contents of the record area and of LINAGE-COUNTER are unchanged, and the following actions take place:

- If the file with which *record-name-1* is associated has a file status data item, its value is set to a value indicating a page boundary violation.
- If a USE procedure is explicitly or implicitly specified for the file associated with *record-name-1*, that declarative procedure is executed.
- If a USE procedure is not explicitly or implicitly specified for the file associated with *record-name-1*, control is transferred to the next executable statement.

INVALID KEY and NOT INVALID KEY Phrases

INVALID KEY *imperative-statement-1*

NOT INVALID KEY *imperative-statement-2*

The invalid key condition exists under one of the following circumstances:

- When a relative file has random or dynamic access mode and the relative key data item specifies a record which already exists in the file.
- When the access mode is sequential for an indexed file opened in the output mode, and the value of the prime record key is not greater than the value of the prime record key of the previous record, except that, if the DUPLICATES phrase is specified in the RECORD KEY clause of the file control entry, the value of the prime record key may be equal to the value of the prime record key of the previous record.
- When an indexed file is opened in the output or I-O mode, and the value of the prime record key is equal to the value of the prime record key of a record already existing in the file and the DUPLICATES phrase is not specified in the RECORD KEY clause of the file control entry.
- When an indexed 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.
- When an attempt is made to write beyond the externally defined boundaries of the file.

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.

Transfer of control following the successful or unsuccessful execution of a Format 2 WRITE statement depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the WRITE statement. This topic is presented in detail in the discussions of invalid key conditions on pages 5-56 and 5-65.

WRITE Statement Examples

```
WRITE TR-RECORD OF TRANSACTION-FILE.


WRITE PF-RECORD FROM TITLE-LINE
  AFTER ADVANCING PAGE.


WRITE PF-RECORD OF PRINT-FILE
  AFTER ADVANCING CHANNEL-1.


WRITE RF-RECORD FROM DETAIL-LINE
  AFTER ADVANCING TO LINE 10
AT END-OF-PAGE
  ADD 1 TO PAGE-COUNT
END-WRITE.


WRITE DB-RECORD OF DATA-BASE
INVALID KEY PERFORM BAD-KEY-PROCEDURE
END-WRITE.


MOVE 5 TO INVENTORY-KEY.
WRITE INVENTORY-RECORD FROM NEW-INVENTORY-ITEM
INVALID KEY DISPLAY "Key 5 not accepted."
NOT INVALID KEY DISPLAY "Key 5 written."
END-WRITE.
```

Appendix A: Reserved Words

This appendix lists all RM/COBOL reserved words. Some words are reserved only for use in the Debug and Report Writer modules; since these modules are not implemented in this version of RM/COBOL, such words do not appear elsewhere in the syntax formats.

Reserved Words

ACCEPT
ACCESS
ADD
ADDRESS [1]
ADVANCING
AFTER
ALL
ALPHABET [1]
ALPHABETIC
ALPHABETIC-LOWER [1]
ALPHABETIC-UPPER [1]
ALPHANUMERIC [1]
ALPHANUMERIC-EDITED [1]
ALSO [1]
ALTER
ALTERNATE
AND
ANY [1]
ARE
AREA
AREAS
ASCENDING [1]
ASSIGN
AT
AUTHOR

BEEP
BEFORE
BELL [1]
BINARY
BLANK
BLINK
BLOCK
BOTTOM [1]
BY

CALL
CANCEL
CD [1]
CENTURY-DATE [1]
CENTURY-DAY [1]
CF [1]
CH [1]
CHARACTER
CHARACTERS
CLASS [1]
CLOCK-UNITS [1]
CLOSE
COBOL [1]
CODE [1]
CODE-SET
COL [1]
COLLATING
COLUMN [1]
COMMA
COMMON [1]
COMMUNICATION [1]
COMP

[1] This word is not considered reserved if the RM/COBOL (74) 2.0 compatibility option is present in the Compile Command (see the *RM/COBOL User's Guide* for details on this option). In such cases, this word is treated as a user-defined word whenever it occurs in the source program.

COMP-1
COMP-3
COMP-4 [1]
COMP-6
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-3
COMPUTATIONAL-4 [1]
COMPUTATIONAL-6
COMPUTE
CONFIGURATION
CONTAINS
CONTENT [1]
CONTINUE [1]
CONTROL [1]
CONTROLS [1]
CONVERT
CONVERTING [1]
COPY
CORR
CORRESPONDING
COUNT [1]
COUNT-MAX [1]
COUNT-MIN [1]
CURRENCY
CURSOR [1]

DATA
DATA-POINTER [1]
DATE
DATE-AND-TIME [1]
DATE-COMPILED [1]
DATE-WRITTEN
DAY
DAY-AND-TIME [1]
DAY-OF-WEEK [1]
DE [1]
DEBUG-CONTENTS [1]
DEBUG-ITEM [1]
DEBUG-LINE [1]
DEBUG-NAME [1]
DEBUG-SUB-1 [1]
DEBUG-SUB-2 [1]
DEBUG-SUB-3 [1]
DEBUGGING [1]
DECIMAL-POINT
DECLARATIVES
DEFAULT [1]
DELETE
DELIMITED [1]
DELIMITER [1]
DEPENDING
DESCENDING [1]
DESTINATION [1]
DETAIL [1]
DISABLE [1]
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

ECHO
EGI [1]
ELSE
EMI [1]
ENABLE [1]
END
END-ACCEPT [1]
END-ADD [1]
END-CALL [1]
END-COMPUTE [1]
END-DELETE [1]
END-DIVIDE [1]
END-EVALUATE [1]
END-IF [1]
END-MULTIPLY [1]
END-OF-PAGE [1]
END-PERFORM [1]
END-READ [1]
END-RECEIVE [1]

---

[1] This word is not considered reserved if the RM/COBOL (74) 2.0 compatibility option is present in the Compile Command (see the *RM/COBOL User's Guide* for details on this option). In such cases, this word is treated as a user-defined word whenever it occurs in the source program.

END-RETURN [1]
END-REWRITE [1]
END-SEARCH [1]
END-START [1]
END-STRING [1]
END-SUBTRACT [1]
END-UNSTRING [1]
END-WRITE [1]
ENTER [1]
ENVIRONMENT
EOP [1]
EQUAL
ERASE
ERROR
ESCAPE [1]
ESI [1]
EVALUATE [1]
EVERY [1]
EXCEPTION
EXCLUSIVE [1]
EXIT
EXTEND
EXTERNAL [1]

FALSE [1]
FD
FILE
FILE-CONTROL
FILLER
FINAL [1]
FIRST
FIXED [1]
FOOTING [1]
FOR
FROM
FUNCTION [1]
GENERATE [1]
GIVING
GLOBAL [1]
GO
GOBACK [1]
GREATER
GROUP [1]

HEADING [1]
HIGH
HIGH-VALUE
HIGH-VALUES
HIGHLIGHT [1]

I-O
I-O-CONTROL
ID [1]
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE [1]
INITIAL
INITIALIZE [1]
INITIATE [1]
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY

LABEL
LAST [1]
LEADING
LEFT
LENGTH [1]
LESS
LIKE [1]

---

[1] This word is not considered reserved if the RM/COBOL (74) 2.0 compatibility option is present in the Compile Command (see the *RM/COBOL User's Guide* for details on this option). In such cases, this word is treated as a user-defined word whenever it occurs in the source program.

LIMIT [1]
LIMITS [1]
LINAGE [1]
LINAGE-COUNTER [1]
LINE
LINE-COUNTER [1]
LINES
LINKAGE
LOCK
LOW
LOWLIGHT [1]
LOW-VALUE
LOW-VALUES

MEMORY
MERGE [1]
MESSAGE [1]
MODE
MODULES
MOVE
MULTIPLY

NATIVE
NEGATIVE [1]
NEXT
NO
NOT
NULL [1]
NULLS [1]
NUMBER [1]
NUMERIC
NUMERIC-EDITED [1]

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL [1]
OR
ORDER [1]
ORGANIZATION
OTHER [1]
OUTPUT
OVERFLOW

PACKED-DECIMAL [1]
PADDING [1]
PAGE
PAGE-COUNTER [1]
PERFORM
PF [1]
PH [1]
PIC
PICTURE
PLUS [1]
POINTER [1]
POSITION
POSITIVE [1]
PRINTING [1]
PROCEDURE
PROCEDURES [1]
PROCEED
PROGRAM
PROGRAM-ID
PROMPT
PURGE [1]

QUEUE [1]
QUOTE
QUOTES
RANDOM
RD [1]
READ
RECEIVE [1]
RECORD
RECORDING [1]
RECORDS
REDEFINES
REEL

---

[1] This word is not considered reserved if the RM/COBOL (74) 2.0 compatibility option is present in the Compile Command (see the *RM/COBOL User's Guide* for details on this option). In such cases, this word is treated as a user-defined word whenever it occurs in the source program.

REFERENCE [1]
REFERENCES [1]
RELATIVE
RELEASE [1]
REMAINDER
REMARKS [1]
REMOVAL [1]
RENAMES
REPLACE [1]
REPLACING
REPORT [1]
REPORTING [1]
REPORTS [1]
RERUN [1]
RESERVE
RESET [1]
RETURN [1]
RETURN-CODE [1]
RETURNING [1]
REVERSE
REVERSE-VIDEO [1]
REVERSED [1]
REWIND
REWRITE
RF [1]
RH [1]
RIGHT
ROUNDED
RUN

SAME
SCREEN [1]
SD [1]
SEARCH [1]
SECTION
SECURE [1]
SECURITY
SEGMENT [1]
SEGMENT-LIMIT [1]
SELECT
SEND [1]
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT [1]
SORT-MERGE [1]
SOURCE [1]
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2 [1]
START
STATUS
STOP
STRING [1]
SUB-QUEUE-1 [1]
SUB-QUEUE-2 [1]
SUB-QUEUE-3 [1]
SUBTRACT
SUM [1]
SUPPRESS [1]
SYMBOLIC [1]
SYNC
SYNCHRONIZED
TAB
TABLE [1]
TALLYING
TAPE [1]
TERMINAL [1]
TERMINATE [1]
TEST [1]
TEXT [1]
THAN
THEN [1]
THROUGH

---

[1] This word is not considered reserved if the RM/COBOL (74) 2.0 compatibility option is present in the Compile Command (see the *RM/COBOL User's Guide* for details on this option). In such cases, this word is treated as a user-defined word whenever it occurs in the source program.

THRU
TIME
TIMES
TO
TOP [1]
TRAILING
TRUE [1]
TYPE [1]

UNIT
UNLOCK
UNSTRING [1]
UNTIL
UP
UPDATE
UPON [1]
USAGE
USE
USING

VALUE
VALUES
VARIABLE [1]
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

The DERESERVE keyword of the COMPILER-OPTIONS configuration record, which is described in the "Configuration" chapter of the *RM/COBOL User's Guide*, can be used to make a reserved word a user-defined word whenever it occurs in the source program, but then the language feature provided by the construct in which the word appears is not available for programs compiled with that particular configuration setting.

[1] This word is not considered reserved if the RM/COBOL (74) 2.0 compatibility option is present in the Compile Command (see the *RM/COBOL User's Guide* for details on this option). In such cases, this word is treated as a user-defined word whenever it occurs in the source program.

Context-Sensitive Words

The words listed in Table A-1 are context-sensitive words and are reserved in the specified language construct or context. If a context-sensitive word is used where the context-sensitive word is permitted in the general format, the word is treated as a keyword; otherwise, it is treated as a user-defined word.

Table A-1: Context-Sensitive Words

Context-Sensitive Word	Language Construct or Context
AUTO	screen description entry
AUTOMATIC	LOCK MODE clause
BACKGROUND	screen description entry
BACKGROUND-COLOR	screen description entry
CASE-INSENSITIVE	LIKE relational-operator
CASE-SENSITIVE	LIKE relational-operator
CYCLE	EXIT statement (Format 3)
EOL	ERASE clause in screen description entry and ERASE phrase in ACCEPT and DISPLAY statements
EOS	ERASE clause in screen description entry and ERASE phrase in ACCEPT and DISPLAY statements
FOREGROUND	screen description entry
FOREGROUND-COLOR	screen description entry
FULL	screen description entry
MANUAL	LOCK MODE clause
MULTIPLE	LOCK MODE clause and I-O-CONTROL paragraph
PARAGRAPH	EXIT statement (Format 4)
PREVIOUS	READ statement (Format 1)
REQUIRED	screen description entry
TRIMMED	LIKE relational-operator
UNDERLINE	screen description entry
YYYYDDD	FROM DAY phrase of ACCEPT statement (Format 2)
YYYYMMDD	FROM DATE phrase of ACCEPT statement (Format 2)

Except for EOL and EOS, the words in Table A-1 are not considered to be context-sensitive words if the RM/COBOL (74) 2.0 compatibility option is present in the Compile Command (see the *RM/COBOL User's Guide* for details on this option). When that option is present, the words in this list, other than EOL and EOS, are treated as user-defined words whenever they occur in the source program.

The DERESERVE keyword of the COMPILER-OPTIONS configuration record, which is described in the "Configuration" chapter of the *RM/COBOL User's Guide*, can be used to make a context-sensitive word a user-defined word whenever it occurs in the source program, but then the language feature provided by the construct in which the word appears is not available for programs compiled with that particular configuration setting.

Special Symbols

The following lists all special symbols except those used in PICTURE character-strings.

.	(
;	"
)	+
'	*
-	**
/	>
=	>=
<	<=
,	= =
:	*>

Nonreserved System-Names

Table A-2 contains system-names that are used in the SPECIAL-NAMES paragraph of the Environment Division to define mnemonic-names and alphabet-names. They are not reserved.

Table A-2: System-Names Used in the SPECIAL-NAMES Paragraph

System-Name	Meaning
C01	Channel 1 ADVANCING for SEND, WRITE statements.
C02	Channel 2 ADVANCING for SEND, WRITE statements.
C03	Channel 3 ADVANCING for SEND, WRITE statements.
C04	Channel 4 ADVANCING for SEND, WRITE statements.
C05	Channel 5 ADVANCING for SEND, WRITE statements.
C06	Channel 6 ADVANCING for SEND, WRITE statements.
C07	Channel 7 ADVANCING for SEND, WRITE statements.
C08	Channel 8 ADVANCING for SEND, WRITE statements.
C09	Channel 9 ADVANCING for SEND, WRITE statements.
C10	Channel 10 ADVANCING for SEND, WRITE statements.
C11	Channel 11 ADVANCING for SEND, WRITE statements.
C12	Channel 12 ADVANCING for SEND, WRITE statements.
CONSOLE	Standard system input-output device (primary terminal).
EBCDIC	Alphabet code-name for EBCDIC as defined by IBM.
SWITCH-1	Switch 1, switch-status conditions and SET statement.
SWITCH-2	Switch 2, switch-status conditions and SET statement.
SWITCH-3	Switch 3, switch-status conditions and SET statement.
SWITCH-4	Switch 4, switch-status conditions and SET statement.
SWITCH-5	Switch 5, switch-status conditions and SET statement.
SWITCH-6	Switch 6, switch-status conditions and SET statement.
SWITCH-7	Switch 7, switch-status conditions and SET statement.
SWITCH-8	Switch 8, switch-status conditions and SET statement.
SYSIN	Standard system input device or file.
SYSOUT	Standard system output device or file.
UPSI-0	Switch 1, switch-status conditions and SET statement.
UPSI-1	Switch 2, switch-status conditions and SET statement.
UPSI-2	Switch 3, switch-status conditions and SET statement.
UPSI-3	Switch 4, switch-status conditions and SET statement.
UPSI-4	Switch 5, switch-status conditions and SET statement.
UPSI-5	Switch 6, switch-status conditions and SET statement.
UPSI-6	Switch 7, switch-status conditions and SET statement.
UPSI-7	Switch 8, switch-status conditions and SET statement.

Table A-3 contains system-names that are used in the FILE-CONTROL paragraph of the Environment Division to specify a device type for files. They are not reserved.

Table A-3: System-Names for Device Types

System-Name	Meaning
CARD-PUNCH	Any sequential output-only device.
CARD-READER	Any sequential input-only device.
CASSETTE	Any sequential input and output device.
CONSOLE	Any sequential input and output device.
DISC	Any mass storage device.
DISK	Any mass storage device.
KEYBOARD	Any sequential input-only device.
LISTING	Any sequential print output device.
MAGNETIC-TAPE	Any sequential input and output device.
PRINT	Any sequential print output device.
PRINTER	Any sequential print output device.
PRINTER-1	Any sequential print output device.
SORT-WORK	Any input and output device for temporary work files (declares file to be a SORT-MERGE file).

Table A-4 contains system-names that are used in the FILE-CONTROL paragraph of the Environment Division to specify the record delimiting technique for sequential files. They are not reserved.

Table A-4: System-Names for Record Delimiting Techniques

System-Name	Meaning
BINARY-SEQUENTIAL	Binary sequential.
LINE-SEQUENTIAL	Line sequential.

Table A-5 contains system-names that are used in the file description entry of the Data Division to specify label information for files. They are not reserved.

Table A-5: System-Names for Labels

System-Name	Meaning
FILE-ID	Specifies file access name.

Table A-6 contains system names that are used as color-names in the screen description entry to specify foreground and background colors. They are not reserved.

Table A-6: System-Names for Colors

Color-Name	Color Integer	Meaning
BLACK	0	The color black.
BLUE	1	The color blue.
GREEN	2	The color green.
CYAN	3	The color cyan.
RED	4	The color red.
MAGENTA	5	The color magenta.
BROWN	6	The color brown.
WHITE	7	The color white.

Appendix B: Compiler Messages

This appendix lists the informational, warning and error messages that may be generated during compilation. These classes of messages are defined as follows:

1. **I** indicates an information-only message. Information messages often follow a warning or error message to provide additional information.
2. **W** indicates a warning. Warning messages are generated when an error occurs during compilation that does not interrupt compilation and that will not prevent program execution.
3. **E** indicates a severe error. Error messages are generated if the error detected during compilation may cause the program to fail during execution.

Compiler Messages

Italics indicate text replaced by compiler generated values.

0001: I Data-name specified in DATA RECORDS clause is: *data-name-1*

Indicates the data-name of the particular data record that is the subject of the previous summary error message.

0002: I Previous diagnostic message occurred at line *line-number-1*

Provides error threading facilities by pointing to the line location of errors generated during compilation. Only the text of the message is printed.

0003: I Above message caused by line *line-number-1*

Indicates the approximate line number of the first occurrence of the summary error message printed just prior to this message.

0004: I Data-name specified in RECORD KEY clause is: *data-name-1*

Indicates the data-name of the particular record key that is the subject of the previous summary error message.

0005: I Scan resumed.

Scanning was suppressed at the previous error and resumes at the indicated point in the source program.

0006: I (scan suppressed).

This message is printed following any error messages that cause the compiler to suspend source scanning. Only the text of the message is printed.

0007: I Data-name specified in KEY phrase of OCCURS clause is: *data-name-1*

Indicates the data-name for the particular table key that is the subject of the previous summary message.

0008: I ALPHABET literal phrase specifies duplicate character for alphabet-name: *alphabet-name-1*

The alphabet-name is defined with a literal phrase which lists a duplicate character, and the alphabet-name was used in a context that does not allow such a definition. A prior error message indicates how the alphabet-name was used.

0009: I First duplicate character occurs at position *position-number-1* [= *character-value-1*]

An alphabet-name has one or more duplicate characters defined, and the alphabet-name was used in a context that does not allow such a definition. The first or only duplicated character is included in this message. Informative message 8 is always produced prior to this message to provide the alphabet-name.

0015: W Configured binary allocation sizes not supported by specified version of runtime.

The indicated data description entry describes a binary data item with a number of digits such that the configured allocation size for that many digits results in a conflict with the maximum object version specified for the compilation. Binary allocation sizes other than two, four, or eight, or sixteen bytes require at least object version 8. A binary allocation size of sixteen requires at least object version 7. When this warning occurs, the compiler allocates the data item with a size compatible with the object version specified.

0016: W Configured binary allocation sizes do not support the precision specified by the PICTURE character-string.

The indicated data description entry describes a binary data item with a number of digits such that no configured allocation size supports that many digits. When this warning occurs, the compiler uses the traditional RM allocation scheme of two, four, eight, or sixteen bytes depending on the number of digits in the data item.

0017: W Signed literal is associated with unsigned data item; absolute value of literal used.

A signed literal is associated with an unsigned data item. For example, a signed literal is the sending item in a MOVE statement where one or more of the receiving items is an unsigned data item. Since unsigned data items always receive the absolute value of any sending item, the sign in the literal is extraneous and may indicate a program logic error. This warning may indicate that the description of the unsigned data item should be changed to that of a signed data item by including an S symbol in its PICTURE character-string.

0018: W Length of literal associated with THRU or ALSO phrase of ALPHABET clause exceeds one character.

More than one character was given for a literal in the ALPHABET clause. It is assumed that each of the characters of the literal were meant to be listed individually in the order given in the source program.

0019: W Level-number 01 or 77 must start in area A of source program.

Level-number 01 or 77 is found in area B. These level-numbers should be in area A, and are treated as if they appeared in area A.

0020: W Record associated with CD entry has wrong size.

A record description entry following an input CD entry implicitly redefines the record area and must be 87 characters in length. A record description entry following an input I-O entry implicitly redefines the record area and must be 33 characters in length. Record entries for output may vary in length, depending on the DESTINATION TABLE OCCURS clause. However, all record entries within a single output CD entry must be the same length.

0021: W CD entry needs more data-names.

Not all 11 data-names for Option 2 of the communication description entry for input or all 6 data-names for Option 2 of the communication description entry for I-O have been listed. Data entries will be used in the order listed.

0022: W Separator period needed to end COPY or REPLACE statement.

A COPY or REPLACE statement is missing its closing period. A closing period is assumed.

0023: W CURRENCY SIGN literal length exceeds one character.

The literal specified in the CURRENCY SIGN clause is longer than one character in length. Only the first character will be used.

0024: W Header or level indicator is in wrong order within Data Division.

The indicated Data Division division header, section header, paragraph header, or level indicator is not in the required order for a COBOL source program. Scanning continues without regard to proper order.

0025: W Literal length must not exceed one character.

A character type operand (INSPECT . . . CHARACTERS or ACCEPT . . . PROMPT) specifies more than one character. For ACCEPT . . . PROMPT, only the first character will be used. For INSPECT . . . CHARACTERS, the entire operand will be used.

0026: W Declarative procedure refers to nondeclarative procedure: *procedure-name-1*

A procedure-name specified in the declaratives is not defined in the declaratives. Standard COBOL does not allow references from the declaratives to the imperatives. If later defined, the procedure-name reference will be allowed by RM/COBOL and will execute correctly.

0027: W DEPENDING ON phrase expected in variable occurrence OCCURS clause.

When Format 2 of the OCCURS clause is used, it is expected that the DEPENDING ON phrase will also be included in the clause.

0028: W Header is in wrong order within Environment Division.

The indicated Environment Division division header, section header, or paragraph header is not in the required order for a COBOL source program. Scanning continues without regard to proper order.

0029: W DATA RECORDS data-name not defined for file: *file-name-1*

The DATA RECORD/RECORDS clause in the file description entry (FD) for the indicated file lists a data-name which is not defined as a level 01 record-name of the file.

0030: W PADDING CHARACTER literal or data item length exceeds one character {: *file-name-1* / .}

The data-name or literal specified in the PADDING clause should be one character in length. Only the first character of the specified operand is used.

0031: W VALUE OF LABEL data-name is not defined in Working-Storage Section for file: *file-name-1*

The file label data item of the indicated file-name is not defined in the Working-Storage Section as required by the standard. There is no effect on the object program.

0032: W RECORD KEY data item must not be variable size group for file: *file-name-1*

The record key data-name refers to a data item that is defined as variable in length.

0033: W Records of sort-merge file are too small for USING file or too large for GIVING file.

The record size of the indicated file-name is not appropriate for the context. In a SORT or MERGE statement, the maximum record size of a USING file is greater than the maximum record size of the sort-merge file or the maximum record size of the sort-merge file is greater than the maximum record size of the GIVING file; records will be truncated during the sort or merge operation if the actual record length is greater than the maximum record length of the sort-merge or GIVING file.

0034: W RECORD DEPENDING data item must be unsigned integer for file: *file-name-1*

The numeric data item in the DEPENDING ON phrase of the RECORD clause in the file description entry for the indicated file-name is defined with a sign.

0035: W RELATIVE KEY data item must be unsigned integer for file: *file-name-1*

The relative key declared for the indicated file-name is a signed numeric integer. The standard requires an unsigned numeric integer. The program may be executed, but negative values, if they occur, may cause undesired results.

0036: W FILE STATUS data-name must not be defined in File Section for file: *file-name-1*

The file status data item declared for the indicated file-name is defined in the File Section of the Data Division. The standard rules against this situation. The program will execute, but unpredictable results may occur if, for example, the file status data item is defined within the record area associated with the file.

0037: W Clause conflicts with VALUE clause specified for group.

The indicated data description clause is wrong because a containing group has a VALUE IS clause.

0038: W GO TO, STOP RUN, or GOBACK must be last statement in sequence of imperative statements.

The imperative sequence contains a GOBACK, GO TO or STOP RUN statement which is not the last statement in the sequence. A GOBACK, GO TO or STOP RUN statement must be followed by a period separator or, if contained within another statement, a scope terminator of the containing statement.

0039: W Indicator area contains wrong character.

The nonblank character in the indicator area (column 7) is not an * (comment), / (new page comment), – (continuation), or D (debug line). The character is treated as a blank.

0040: W Integer has value that exceeds maximum permitted for this use.

The integer indicated requires more than 16 bits to represent its value. RM/COBOL is limited to 16-bit words for the representation of some integer values. The value 65535 is used.

0041: W LABEL RECORDS ARE OMITTED and VALUE OF clause must not be specified in same file description.

The VALUE OF and LABEL RECORDS OMITTED clauses are specified for the same file. There is no effect on the object program.

0042: W Unsigned integer expected in MEMORY SIZE clause. Nonnumeric or signed literal is not permitted here.

The literal specified in the MEMORY clause of the OBJECT-COMPUTER paragraph is not unsigned numeric. There is no effect on the object program.

0043: W Unsigned integer expected in MEMORY SIZE clause. Noninteger literal is not permitted here.

The literal specified in the MEMORY clause of the OBJECT-COMPUTER paragraph is not an integer. There is no effect on the object program.

0044: W Repeated period space separator is not permitted.

A period space separator is repeated where only one period space separator is allowed. The unneeded separator is ignored by the compiler.

0045: W Imperative statement expected but scope terminator was found. CONTINUE statement assumed.

An imperative statement is required at the indicated source location, but a scope terminator was specified instead. The compiler assumes a CONTINUE statement was intended.

0046: W Repeated phrase in ACCEPT or DISPLAY statement is not permitted.

An option phrase is specified more than once or the default option has been specified in violation of syntactic rules. The indicated or later occurrence is ignored.

0047: W PERFORM independent segment THRU fixed segment is not permitted.

The EXIT paragraph of the performed procedure (or procedures) is in a fixed segment (segment number less than 50) and the PERFORM statement is in an independent segment. Only sections or paragraphs wholly contained in the fixed segment or wholly contained in the same independent segment should be used.

0048: W PERFORM exit procedure ends with unconditional transfer of control: *procedure-name-1*

A procedure specified as the exit of a PERFORM statement contains an unconditional GOBACK, GO TO or STOP RUN statement as its last statement. Therefore, the procedure cannot reach its exit so as to return control to the controlling PERFORM statement. PERFORM statements which reference such a procedure as the exit procedure are equivalent to a GO TO statement referencing the same entry procedure as specified by the PERFORM statement.

0049: W Procedure-name expected in area A.

A procedure-name is required in area A because of a preceding section header.

0050: W Procedure-name contains wrong character.

A procedure-name contains a decimal point. The decimal point character is ignored.

0051: W EXTERNAL clause requires specification of device-name in ASSIGN clause. DISK assumed.

The ASSIGN clause for a sequential organization file omitted the device-name or specified an unknown device-name and the file is described as EXTERNAL. Since other programs, unknown to this compilation, may access the external file in the I-O mode, the compiler must assume a mass storage device. If a non-mass storage device is intended, specify the appropriate device-name in the ASSIGN clause.

0052: W Space separator expected.

A literal and user-defined word have no separator between them.

0053: W Space character expected after punctuation character.

A comma or semicolon character occurs in the source program without a following space. The comma or semicolon is treated as if the space was present.

0054: W Sort-merge file control entry must contain only SELECT and ASSIGN clauses.

A sort-merge file is declared with file control clauses that are not allowed. The clauses are ignored unless they specify illegal options (for example, nonsequential organization).

0055: W ASCENDING or DESCENDING phrase expected. ASCENDING assumed.

The ASCENDING or DESCENDING key comparison is omitted or misspelled. If simply omitted, ASCENDING will be assumed. If misspelled, a syntax error will also occur.

0056: W TIMES, UNTIL, and VARYING phrases are nonstandard in SORT or MERGE statement.

Non-standard phrases are specified in the INPUT or OUTPUT procedure declaration of a SORT or MERGE statement.

0057: W User-defined word length exceeds 30 characters.

A user-defined word is longer than 30 characters in length. The entire word is used, up to a length of 120 characters, during compilation. Data-names and file-names with the external attribute are truncated to 30 characters in the object program.

0058: W Repeated file-name or open mode in USE statement is not permitted.

A multiple USE declarative exists for the indicated file-name or open mode. When multiple USE declaratives are declared, the last one declared is in effect for the object program, except that a USE declarative for a file-name will take precedence over a USE declarative for an open mode.

0059: W Value of numeric literal in VALUE clause exceeds capacity of PICTURE character-string.

The numeric literal specified in the VALUE clause for a numeric data item is incorrect for initialization of the data item as described by its PICTURE character-string but is within the range of values allowed by the data item. Truncation of nonzero low-order digits was required; or, for a BINARY, COMP-1 or COMP-4 usage data item, more digits were specified than allowed by the PICTURE character-string but the value can still be expressed within the number of bytes allocated for the data item.

0060: W Verb must start in area B of source program.

A verb was found in area A of the source program. The verb is treated as if it occurred in area B.

0061: W Clause must start in area B of source program.

A clause begins in area A of a source record. The clause is treated as if it began in area B.

0062: W Pair of delimiting quotes are not same character.

A hexadecimal literal is not delimited by a matched pair of single or double quotation marks. The compiler assumes that the single or double quotation mark found, even though it does not match the beginning quotation mark, was intended as the ending delimiter for the hexadecimal literal.

0063: W Phrase is not valid for data type being accepted or displayed.

The indicated ACCEPT or DISPLAY option is not allowed for the operand being accepted or displayed. For example, the CONVERT option requires a numeric or edited operand. The option is ignored.

0064: W Phrase is valid only for USAGE IS DISPLAY operand.

One or more of the ACCEPT or DISPLAY options are not allowed for nondisplay (computational) numeric operands. For DISPLAY of a nondisplay data item, the CONVERT phrase is required and is assumed. For ACCEPT of a nondisplay data item, the ECHO phrase is not allowed and is ignored unless the UPDATE phrase is also present.

0065: W Integer value must not be equal to zero.

An integer with the value zero is not allowed in the indicated context. The program may or may not execute correctly. For example, if an index-name is set to the value zero it will contain a value that is not valid for subscripting but a subsequent SET . . . UP BY 1 statement will cause the index-name to contain a valid value for the first occurrence.

0066: W Neither GREATER OR EQUAL (>=) nor LESS THAN OR EQUAL (<=) may be preceded by NOT.

The reserved word NOT should not be used with the relational operators >= and <= or with their spelled-out equivalents. Such cases are treated as < and >, respectively.

0067: W RECORD DELIMITER clause specified with fixed-length records for file: *file-name-1*

A RECORD DELIMITER clause with the STANDARD-1 option has been specified for a file whose records are not variable length.

0068: W Repeated character in CLASS clause is not permitted.

A character has been specified more than once in a CLASS clause. Specifying the same character more than once in a CLASS clause is redundant.

0069: W Nonconforming nonstandard language element found in statement, clause, or header. RM extension to COBOL.

The indicated language element is not defined in the standard COBOL language. It is an extension that is defined and supported by Liant, but which may not be supported in other COBOL dialects. This message is only generated when the flagging of extensions is requested by specification of a compiler option.

0070: W Nonconforming standard language element found in statement, clause, or header.

The indicated language element is defined in the standard COBOL language but at a level above the requested level. This message is only generated when the flagging of COBOL subsets or optional modules is requested by specification of a compiler option.

0071: W Obsolete language element found in statement, clause, or header.

The indicated language element is defined in the standard COBOL language, but has been designated for deletion from the standard language in a future revision. This message is only generated when the flagging of obsolete elements is requested by specification of a compiler option.

0072: W Pseudo-text delimiter must not be continued.

The two characters of a pseudo-text delimiter should be contiguous within the same source record. They are treated as if they were contiguous.

0073: W Pseudo-text operand expected for REPLACE statement.

The two operands of a BY phrase in a REPLACE statement should be pseudo-text operands. Operands that are not written as pseudo-text operands are treated by the compiler like the nonpseudo-text operands of a REPLACING phrase in a COPY statement.

0074: W Statement cannot be executed because preceding statement transfers control.

The indicated statement can never be executed because it is immediately preceded by another statement that transfers control unconditionally to another statement in the program.

0075: W Data item containing file access name must have fixed length for file: *file-name-1*

The data-name specified in the ASSIGN clause or VALUE OF FILE-ID clause for the indicated file-name is a variable length data item (that is, a group that contains a data item described with the OCCURS . . . DEPENDING clause). The maximum length of the group will be used to resolve the file access name for the file. The value of the data item specified in the DEPENDING ON phrase of the OCCURS clause will not be used to determine the length of the group when used for this purpose.

0076: W Device-name is not supported by this implementation.

The indicated word is not recognized as a valid device-name known to this implementation. An unspecified device type is assumed. The unspecified device type is changed to mass storage if any clause (for example, ORGANIZATION IS INDEXED) or statement (for example, OPEN I-O) referencing the file so requires. Otherwise, the file may reside on either mass storage or non-mass storage media.

0077: W Contained program has wrong structure.

The indicated syntax is not allowed in a program nested within another program. Clauses which typically affect the remainder of the program are ignored. Declarations of mnemonic-names, alphabet-names, symbolic-characters and class-names are accepted and used for the remainder of the compilation of the separately compiled program which contains the nested program, including programs not nested within the program which was diagnosed with this warning.

If nested programs were not desired, add the END PROGRAM headers or use the Compile Command Option (see Chapter 6, *Compiling*, of the *RM/COBOL User's Guide*) to set the object version level to 1 or 2. Since nested programs are not supported prior to version 3, restricting the object version level to 1 or 2 causes the compiler to assume that—even in the absence of END PROGRAM headers—the source file contains a sequence of source programs rather than nested source programs.

0078: W END PROGRAM header expected.

An END PROGRAM header is required because a nested program has been scanned and the matching END PROGRAM header has not been found, either for the nested program itself or for each of its containing programs.

If nested program were not desired, use the Compile Command Option to set the object version level to 1 or 2 (see Chapter 6, *Compiling*, of in the *RM/COBOL User's Guide*). This causes the compiler to assume that—even in the absence of END PROGRAM headers—the source file contains a sequence of source programs rather than nested source programs.

0079: W Program-name is not unique within this separately compiled program.

A nested program specifies the same program-name as another program within the separately compiled program containing that nested program. When source programs are nested, a particular program-name may only occur once in the PROGRAM-ID paragraph of any program contained in the separately compiled program.

0080: E Figurative constant preceded by ALL is not permitted.

Use of the “ALL” form of a figurative constant is not allowed in the indicated context.

0081: E Numeric literal in ALPHABET clause exceeds 256, maximum number of characters in native character set.

The integer used in the ALPHABET clause of the SPECIAL-NAMES paragraph must represent an ordinal position in the native character set. The number of characters in the native character set is 256.

0082: E Alphabet-name associated with COLLATING SEQUENCE clause or phrase must not have duplicate character.

A character has been specified more than once in the ALPHABET clause that defines the alphabet-name specified in the COLLATING SEQUENCE clause of the File-Control entry or in the COLLATING SEQUENCE phrase of the SORT and MERGE statements. Since a character can have only one collating position, a character must not be repeated in the definition of an alphabet-name specified as a collating sequence. Refer to the “ASCII Position” and “U.S. Character” columns in Appendix J, *Code-Set Translation Tables*, of the *RM/COBOL User’s Guide*. Informational messages 8 and 9 (see page B-2) are generated at the end of the program listing to provide the alphabet-name and duplicated character.

0083: E Class-name in CLASS clause is not unique.

The indicated user-defined word has already been defined for some other purpose and cannot be used to define a class-name.

0084: E Class-name in class condition is not defined by CLASS clause.

The context suggests that a class-name is intended at the indicated position, but the specified user-defined word is undefined.

0085: E Alphabet-name in ALPHABET clause is not unique.

The indicated user-defined word has already been defined for some other purpose and cannot be used to define an alphabet-name.

0086: E Alphabet-name expected.

The context requires an alphabet-name, but the indicated user-defined word is not an alphabet-name.

0087: E Alphabet-name is not defined by ALPHABET clause.

The indicated context requires an alphabet-name, but the given user-defined word is undefined.

0088: E Wrong code-name in ALPHABET clause.

An unrecognized type is given in the ALPHABET clause of the SPECIAL-NAMES paragraph. Valid alphabet types are STANDARD-1, EBCDIC, NATIVE or a literal phrase.

0089: E ALTER in nondeclarative procedure must not refer to declarative procedure: *procedure-name-1*

An ALTER statement in the nondeclaratives region is wrong because the procedure-name of the procedure to be altered, the paragraph containing the alterable GO TO statement, refers to a declarative procedure.

0090: E ALTER of independent segment must be in same independent segment.

An ALTER statement is wrong because the procedure-name of the procedure to be altered, the paragraph containing the alterable GO TO statement, refers to a procedure defined in an independent segment which has a different segment number than the segment containing the ALTER statement.

0091: E ALTER must refer to alterable paragraph that contains only a GO TO sentence.

An ALTER statement is wrong because the procedure-name of the procedure to be altered, the paragraph containing the alterable GO TO statement, does not refer to a paragraph containing only a single Format 1 GO TO statement.

0092: E ALTER refers to procedure-name that is not unique: *procedure-name-1*

An ALTER statement is wrong because the procedure-name of the procedure to be altered, the paragraph containing the alterable GO TO statement, refers to two or more procedures. Qualification of the paragraph-name by its section-name is required to yield a unique reference.

0093: E ALTER refers to procedure-name that is not defined: *procedure-name-1*

An ALTER statement is wrong because the procedure-name of the procedure to be altered, the paragraph containing the alterable GO TO statement, refers to a procedure which is undefined. The procedure-name may be incorrectly qualified.

0094: E GO TO statement omits procedure-name. No ALTER statement found for paragraph: *procedure-name-1*

A GO TO statement with the procedure-name omitted is not the object of any ALTER statement and, therefore, can never be executed successfully.

0095: E Continuation of nonnumeric literal must begin with quotation mark.

A nonnumeric literal is continued but does not have the required opening quotation mark on the continuation line.

0096: E Nonnumeric literal expected.

The context requires a nonnumeric literal.

0097: E Nonnumeric literal length exceeds 2047 characters.

A nonnumeric literal greater than 2047 characters in length is specified by the source program. Standard COBOL requires support of literals up to 160 characters in length. RM/COBOL supports literals up to 2047 characters in length.

0098: E Nonnumeric literal must end with quotation mark.

A nonnumeric literal is not continued and does not have the required closing quotation mark.

0099: E Header or level indicator expected in area A of source program.

Context requires an entry in area A at the indicated point in the source program.

0100: E Arithmetic expression has wrong combination of operands and symbols.

The syntax of the arithmetic expression is wrong. The permissible combinations of variables, numeric literals, arithmetic operators and parentheses are given in Table 5-1 on page 5-21.

0101: E ASSIGN clause required in file control entry.

No ASSIGN clause was found in the file control entry which begins with a SELECT clause and ends with a period. The ASSIGN clause is required in a file control entry.

0102: E AT END phrase required in RETURN statement.

The AT END clause is required in a RETURN statement, but was not found.

0103: E BLANK WHEN ZERO clause requires elementary numeric or numeric edited data item with USAGE IS DISPLAY.

The BLANK WHEN ZERO clause is specified in a data description entry in conflict with other clauses specified in the same entry.

0104: E Repeated clause in CD entry is not permitted.

A clause in the communication description entry has been specified more than once.

0105: E INITIAL clause must not be specified in program having USING or GIVING phrase in Procedure Division header.

The INITIAL clause of the communication description entry may not be used in a program that specifies the USING or GIVING phrases in the Procedure Division header.

0106: E Repeated INITIAL clause in program is not permitted.

More than one communication description entry with the INITIAL clause has been specified in the source program. Only one CD FOR INITIAL INPUT or one CD FOR INITIAL I-O is allowed in a program.

0107: E Input CD entry has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the Format 1 (FOR INPUT) communication description entry as given in the source program.

0108: E I-O CD entry has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the Format 3 (FOR I-O) communication description entry as given in the source program.

0109: E Output CD entry has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the Format 2 (FOR OUTPUT) communication description entry as given in the source program.

0110: E Cd-name is not unique.

A cd-name has previously been defined and cannot be used again.

0111: E Cd-name of input or I-O CD entry expected.

An INPUT or I-O cd-name must be specified in the context of the statement as given in the source program. An INPUT cd-name is required with ACCEPT MESSAGE COUNT, ENABLE INPUT and DISABLE INPUT statements. An INPUT or I-O cd-name is required with RECEIVE statements.

0112: E Cd-name of output or I-O CD entry expected.

An OUTPUT or I-O cd-name must be specified in the context of the statement as given in the source program. An OUTPUT cd-name is required with ENABLE OUTPUT and DISABLE OUTPUT statements. An OUTPUT or I-O cd-name is required with SEND statements.

0113: E Cd-name is not permitted here.

The context does not allow a cd-name reference.

0114: E Associated CD entry contains error.

There is an error in the communication description entry associated with the indicated cd-name.

0115: E Cd-name is not defined by CD entry in program.

The cd-name specified in a SEND, RECEIVE, ACCEPT, ENABLE or DISABLE statement is not defined in the current program. A communication description entry is required in the context of the statement indicated in the source program. Only cd-names declared in the Data Division associated with the Procedure Division may be specified in the Procedure Division (that is, cd-names are always local names, never global).

0116: E Length of record associated with CD entry exceeds 65280 characters.

The maximum CD record size has been exceeded. Only the first 65280 characters will be used.

0117: E CD entry needs FOR INPUT, FOR OUTPUT, or FOR I-O clause.

The INPUT, OUTPUT or I-O clause is required in the communication description entry.

0118: E Mnemonic-name in ADVANCING phrase must be associated with channel-name in SPECIAL-NAMES paragraph.

The indicated user-defined word must be identified with a feature-name that is a channel-name in the SPECIAL-NAMES paragraph of the Environment Division.

0119: E Operand data type not permitted for this class condition.

The specified class condition conflicts with the data type of the item being tested. An alphabetic data item may not be specified in the NUMERIC class test. A numeric data item may not be specified in the ALPHABETIC, ALPHABETIC-LOWER, or ALPHABETIC-UPPER class tests.

0120: E CODE-SET clause in FD entry must specify same alphabet as CODE-SET clause in file control entry.

A code-set was previously defined in the file control entry for the indicated file and does not match the code-set in the file description entry. The code-set should be specified only once, but if specified in both the file control entry and file description entry, the specifications must be consistent.

0121: E CODE-SET clause requires all signed data items for file to specify SIGN IS SEPARATE CHARACTER.

A file that is defined with a CODE-SET clause must have a SIGN SEPARATE clause in all signed numeric data descriptions in the record descriptions associated with the file.

0122: E CODE-SET clause requires all data items for file to have USAGE IS DISPLAY.

A file that is defined with a CODE-SET clause must not have any numeric data items defined with a USAGE IS clause except USAGE IS DISPLAY in the record descriptions associated with the file.

0123: E Only one PROGRAM COLLATING SEQUENCE clause is permitted.

More than one PROGRAM COLLATING SEQUENCE clause has been specified in the source program. Only one is allowed.

0124: E COLLATING SEQUENCE clause permitted only in indexed file control entry.

The COLLATING SEQUENCE clause may be specified for indexed organization files only. Relative and sequential file control entries may not include the COLLATING SEQUENCE clause.

0125: E Composite of operands contains more than 30 decimal digits.

The composite of operands specified in the indicated statement contains more than 30 digits. The total integer positions plus the total fractional positions must not exceed 30 for the specified operands. (See the discussion of composite size on page 5-17 for additional information.)

0126: E Associated conditional variable has error in its data description entry.

The condition-name indicated is associated with a conditional variable which has an error in its description.

0127: E User-defined word previously defined for use that does not permit its use as condition-name.

The indicated user-defined word has already been defined and cannot be redefined as a condition-name.

0128: E Condition-name is not permitted here.

The context does not allow a condition-name, but the identifier indicated is that of a condition-name.

0129: E Data description entry for condition-name must specify single value for use in SEARCH ALL statement.

A condition-name specified in a SEARCH ALL statement must have a single value associated with it, but the indicated condition-name has multiple values associated with it.

0130: E Literal in VALUE clause has wrong category for data type of associated conditional-variable.

The value literal specified for a condition-name has a type which conflicts with the type of the associated conditional variable.

0131: E Condition has wrong combination of conditions, logical operators, and parentheses.

The syntax of a conditional expression is incorrect. The syntax given in the source is not that of a relation, class, sign, condition-name or switch-status condition. This syntax error may be caused by failure to follow the separator rules of COBOL which require spaces around the special characters used in relation conditions.

0132: E COPY statement exceeds maximum nesting level of 9 active COPY statements.

The maximum copy nesting level of nine has been exceeded. Only five copy files may be open, but the maximum nesting level of nine can be exceeded when COPY statements are the last statement in a COPY file. In such cases, the COPY file is closed before opening the next COPY file, but the COPY statement is still considered to be nested.

0133: E Text-name or file-name in COPY statement is not accessible to compiler.

The text-name and, optionally, library-name specified in a COPY statement indicated copy text which could not be accessed. The file could not be opened, possibly because the nesting level of five open copy files has been exceeded or because the copy text contains a COPY statement which copies itself, either directly or indirectly.

0134: E CORRESPONDING operand must be group data item not defined with RENAMES clause.

The context requires an identifier of a group data item which satisfies the rules for CORRESPONDING. The identifier indicated is either not a group or is a group with no subordinate named data items (for example, a group defined by RENAMES).

0135: E CORRESPONDING operands have no corresponding numeric data items.

The two groups specified in an ADD CORRESPONDING or SUBTRACT CORRESPONDING statement have no corresponding numeric items. These statements require at least one pair of corresponding numeric items.

0136: E CURRENCY SIGN literal contains wrong character.

The literal specified in the CURRENCY SIGN clause specifies a character which is not allowed for the currency symbol.

0137: E Data description entry has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the data description entry or screen description entry as given in the source program.

0138: E Data Division has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the Data Division as given in the source program.

0139: E User-defined word or literal expected.

The context requires a reference to a data item or literal but the indicated character-string or separator does not reference data.

0140: E Data item described in Linkage Section is not addressable from USING phrase of Procedure Division header.

The indicated identifier refers to a data item or condition-name defined in the Linkage Section of the Data Division but its data-name or the data-name of its conditional variable is not listed in the USING phrase of the Procedure Division header nor is the data item or condition-name defined subordinate to a data-name listed in the USING phrase of the Procedure Division header nor is it a redefinition or rename of such a name. Thus, the identifier refers to a data item which would not be addressable by the program at object time.

Note Message 140 only occurs when the object version is restricted to less than 8. Object version 8 supports the ability to make any Linkage Section data item addressable by use of the Format 5 SET statement. For object version 8 or greater, if a Linkage Section data item is referenced and the base address is never set within the program, message 665 will occur.

0141: E Repeated clause in data description entry is not permitted.

The indicated data description clause is repeated for the same subject.

0142: E User-defined word previously defined for use that does not permit its use as data-name.

The indicated user-defined word is already defined for a purpose that conflicts with its use as a data-name.

0143: E Data item length exceeds 65280 characters.

The indicated data item has a size greater than 65280 characters. Such items may be specified only in a MOVE statement or in the USING phrase of a CALL statement.

0145: E Alphanumeric data item expected.

The context requires an alphanumeric data item.

0146: E Elementary data item expected.

The indicated identifier does not refer to an elementary data item as required by the context in which it is specified. The identifier refers to a group data item and group data items are not allowed in this context.

0147: E Data item with DISPLAY usage expected.

The context requires a data item with DISPLAY usage.

0148: E Data item described with JUSTIFIED clause is not permitted.

The JUSTIFIED clause cannot be used in the data description entry of the data-name specified in the indicated context.

0149: E Numeric integer data item or literal expected.

The context requires a numeric integer data item.

0150: E Numeric or numeric edited data item expected.

The context requires a numeric or a numeric edited data item.

0151: E Numeric data item or literal expected. Numeric edited data item is not permitted here.

The context requires a numeric data item; a numeric edited data item is not allowed.

0152: E Literal in VALUE clause has wrong category for data item described by data description entry.

The literal type specified in the VALUE literal for a data description entry conflicts with the data type of the item as described by other clauses.

0153: E END DECLARATIVES header expected.

An END header was found while scanning the declaratives portion of the Procedure Division, but it was not the END DECLARATIVES header.

0154: E GO TO or ALTER statement in nondeclarative procedure must not refer to declarative procedure: *procedure-name-1*

A GO TO or ALTER statement in the imperatives refers to a procedure-name defined in the declaratives. All GO TO and ALTER statements in the imperatives must refer to procedure-names defined in the imperatives.

0155: E Section header must follow DECLARATIVES header.

The declaratives must begin with a section definition.

0156: E Segment-number in declaratives section header exceeds 49.

A segment-number greater than 49 is given in the declaratives. Independent segments are not allowed in the declaratives. The last valid segment-number is used instead.

0157: E Statement is not permitted in declarative procedure or in GLOBAL declarative procedure.

The indicated statement clashes with the declaratives context in which it is specified. A SORT or MERGE statement is not allowed anywhere in the declaratives portion of the Procedure Division. An EXIT PROGRAM or GOBACK statement is not allowed in a declarative procedure in which the GLOBAL phrase is specified.

0158: E Level-number less than or equal to level-number in previous elementary data description entry expected.

The previous data description entry defined an elementary data item, but the indicated level-number is not less than or equal to the level-number of the previous entry.

0159: E Environment Division has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the Environment Division as given in the source program.

0160: E EQUAL (=) relational operator required in WHEN phrase of SEARCH ALL statement.

The condition specified is not an equal relation. In a SEARCH ALL statement only a condition-name or an equal relation is allowed.

0161: E Data description entry for condition-name must have WHEN SET TO FALSE phrase.

A condition-name cannot be set to false unless the WHEN SET TO FALSE phrase is specified in the data description entry for the condition-name.

0162: E File description entry (FD entry) has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the file description entry as given in the source program.

0163: E BOTTOM data item must be unsigned integer for file: *file-name-1*

The data-name specified in the LINES AT BOTTOM phrase of the LINAGE clause for the indicated file does not refer to an elementary unsigned numeric data item.

0164: E BOTTOM data-name has error in its data description entry for file: *file-name-1*

The data-name specified in the LINES AT BOTTOM phrase of the LINAGE clause for the indicated file has an error in its data description.

0165: E BOTTOM operand must refer to data item for file: *file-name-1*

The data-name specified in the LINES AT BOTTOM phrase of the LINAGE clause for the indicated file is not a valid data item described in the Data Division.

0166: E BOTTOM data item must not be table element for file: *file-name-1*

The data-name specified in the LINES AT BOTTOM phrase of the LINAGE clause for the indicated file cannot be defined with an OCCURS clause.

0167: E BOTTOM data-name is not unique for file: *file-name-1*

The data-name specified in the LINES AT BOTTOM phrase of the LINAGE clause for the indicated file is defined more than once and is not adequately qualified.

0168: E BOTTOM data-name is not defined for file: *file-name-1*

The data-name specified in the LINES AT BOTTOM phrase of the LINAGE clause for the indicated file has not been defined. An elementary unsigned numeric data entry in the Data Division is required.

0169: E BOTTOM data item is wrong linkage item or is not external item for file: *file-name-1*

- The data-name specified in the LINES AT BOTTOM phrase of the LINAGE clause for the indicated file has been defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, nor is it defined subordinate to such a data-name. The data-name should be included as a USING parameter or defined outside the Linkage Section.

  Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

- Or, the indicated file-name names an external file connector but the data-name specified in the LINES AT BOTTOM phrase of the LINAGE clause does not possess the external attribute as required.

0170: E Open mode not permitted for file with CODE-SET clause that specifies alphabet with duplicate character.

A file with a CODE-SET clause specifies an alphabet-name that has a character used more than once. A file opened for any mode other than INPUT cannot refer to an alphabet-name that has a character listed more than once. Refer to the "ASCII Position" and "U.S. Character" columns in Appendix J, *Code-Set Translation Tables*, of the *RM/COBOL User's Guide*, for the exact correlation of ordinal position to native character. Informational messages 8 and 9 (see page B-2) are generated at the end of the program listing to provide the alphabet-name and duplicated character.

0171: E Repeated clause in file control entry is not permitted.

A file control clause is repeated for the same file.

0172: E File control entry has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the file control entry as given in the source program.

0173: E Missing or wrong file description entry for file: *file-name-1*

The file description for the indicated file-name is either missing or has an error. No record area is defined for the file.

0174: E Device-name specified for file-name does not permit this operation.

The device associated with file-name does not allow the indicated operation. The device type is determined by the device-name specified in the ASSIGN clause of the file control entry.

0175: E File access name data item must be alphanumeric for file: *file-name -1*

The category of the data item declared as the file access name in the ASSIGN clause or VALUE OF FILE-ID clause for the indicated file-name is not alphanumeric as required.

0176: E File access name data-name has error in its data description for file: *file-name-1*

The data-name declared as the file access name in the ASSIGN clause or VALUE OF FILE-ID clause for the indicated file-name refers to a data item that has an error in its description.

0177: E File access name data-name must refer to data item for file: *file-name-1*

The data-name declared as the file access name in the ASSIGN clause or VALUE OF FILE-ID clause for the indicated file-name does not refer to a data item as required.

0178: E File access name data item must not be table element for file: *file-name-1*

The data-name declared as the file access name in the ASSIGN clause or VALUE OF FILE-ID clause for the indicated file-name refers to a data item that is described with the OCCURS clause or is subordinate to an item described with the OCCURS clause. Since this would require subscripting, it is not allowed.

0179: E File access name data-name is not unique for file: *file-name-1*

The data-name declared as the file access name in the ASSIGN clause or VALUE OF FILE-ID clause for the indicated file-name refers to two or more data items; the qualification is ambiguous.

0180: E File access name data-name is not defined for file: *file-name-1*

The data-name declared as the file access name in the ASSIGN clause or VALUE OF FILE-ID clause for the indicated file-name is undefined.

0181: E File access name data-name is wrong linkage item for file: *file-name-1*

The data-name declared as the file access name in the ASSIGN clause or VALUE OF FILE-ID clause for the indicated file-name refers to a data item defined in the Linkage Section but is neither specified in the Procedure Division USING phrase nor is it subordinate to an item specified in the Procedure Division USING phrase.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0182: E FOOTING data item must be unsigned integer for file: *file-name-1*

The data-name specified in the FOOTING phrase of the LINAGE clause for the indicated file does not refer to an elementary unsigned numeric data item.

0183: E FOOTING data-name has error in its data description entry for file: *file-name-1*

The data-name specified in the FOOTING phrase of the LINAGE clause for the indicated file has an error in its data description.

0184: E FOOTING operand must refer to data item for file: *file-name-1*

The data-name specified in the FOOTING phrase of the LINAGE clause for the indicated file is not a valid data item described in the Data Division.

0185: E FOOTING data item must not be table element for file: *file-name-1*

The data-name specified in the FOOTING phrase of the LINAGE clause for the indicated file cannot be defined with an OCCURS clause.

0186: E FOOTING data-name is not unique for file: *file-name-1*

The data-name specified in the FOOTING phrase of the LINAGE clause for the indicated file is defined more than once and is not adequately qualified.

0187: E FOOTING data-name is not defined for file: *file-name-1*

The data-name specified in the FOOTING phrase of the LINAGE clause for the indicated file has not been defined. An elementary unsigned numeric data entry in the Data Division is required.

0188: E FOOTING data item is wrong linkage item or is not external item for file: *file-name-1*

- The data-name specified in the FOOTING phrase of the LINAGE clause for the indicated file has been defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, nor is it defined subordinate to such a data-name. The data-name should be included as a USING parameter or defined outside the Linkage Section.

  Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

- Or, the indicated file-name names an external file connector but the data-name specified in the FOOTING phrase of the LINAGE clause does not possess the external attribute as required.

0189: E LABEL RECORDS clause must specify STANDARD or OMITTED option.

The LABEL RECORDS clause specifies an unrecognized label option. The label must be described as STANDARD or OMITTED.

0190: E VALUE OF data-name has error in its data description for file: *file-name-1*

The data-name declared in the VALUE OF clause for the indicated file-name refers to a data item that has an error in its description.

0191: E VALUE OF operand must refer to data item for file: *file-name-1*

The data-name declared in the VALUE OF clause for the indicated file-name refers to a nondata item such as an alphabet-name or condition-name.

0192: E VALUE OF data item must not be table element for file: *file-name-1*

The data-name declared in the VALUE OF clause for the indicated file-name refers to a data item that is described with the OCCURS clause or is subordinate to an item described with the OCCURS clause.

0193: E VALUE OF data-name is not unique for file: *file-name-1*

The data-name declared in the VALUE OF clause for the indicated file-name refers to two or more data items; the qualification is ambiguous.

0194: E VALUE OF data-name is not defined for file: *file-name-1*

The data-name declared in the VALUE OF clause for the indicated file-name is undefined.

0195: E VALUE OF data item is wrong linkage item for file: *file-name-1*

The data-name declared in the VALUE OF clause of the file description entry is defined in the Linkage Section but is not listed in the Procedure Division USING phrase, and is not defined subordinate to such a data-name. The data-name should be included as a USING parameter or defined outside the Linkage Section.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0196: E LINAGE data item must be unsigned integer for file: *file-name-1*

The data-name specified in the LINAGE IS clause does not refer to an elementary unsigned numeric data item.

0197: E LINAGE data-name has error in its data description entry for file: *file-name-1*

The data-name specified in the LINAGE IS clause for the indicated file has an error in its data description.

0198: E LINAGE operand must refer to data item for file: *file-name-1*

The data-name specified in the LINAGE IS clause for the indicated file is not a valid data item described in the Data Division.

0199: E LINAGE data item must not be table element for file: *file-name-1*

The data-name specified in the LINAGE IS clause for the indicated file cannot be defined with an OCCURS clause.

0200: E LINAGE data-name is not unique for file: *file-name-1*

The data-name specified in the LINAGE IS clause for the indicated file is defined more than once and is not adequately qualified.

0201: E LINAGE data-name is not defined for file: *file-name-1*

The data-name specified in the LINAGE IS clause has not been defined. An elementary unsigned numeric data entry in the Data Division is required.

0202: E LINAGE data item is wrong linkage item or is not external item for file: *file-name-1*

- The data-name specified in the LINAGE IS clause has been defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, and is not defined subordinate to such a data-name. The data-name should be included as a USING parameter or defined outside the Linkage Section.

  Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

- Or, the indicated file-name names an external file connector but the data-name specified in the LINAGE IS clause does not possess the external attribute as required.

0203: E Repeated FD or SD entry for file-name is not permitted.

The indicated file-name has already been defined in an FD or SD entry and cannot be defined again.

0204: E Repeated clause in file description entry is not permitted.

The indicated file description clause is repeated for the same file.

0205: E User-defined word previously defined for use that does not permit its use as file-name.

The indicated user-defined word is already defined for some other purpose and cannot be defined as a file-name.

0206: E File-name is not permitted here.

The indicated context does not allow a file-name reference. If the indicated context is the first operand of a REWRITE or WRITE statement, a record-name of a file is required instead of the file-name.

0207: E File-name has error in its file control or file description entry.

The indicated file-name has an error in its description.

0208: E File-name expected.

The context requires a file-name.

0209: E File-name is not defined by file control entry.

The indicated file-name is not defined. This includes qualification errors such as an attempt to qualify a file-name.

This error may also indicate that the file-name is defined outside the current program, but is wrong for one of these reasons: the file-name is not global; the file-name is global but is not defined in a program which contains the current program; or a file-name described in the same program is required in this context.

0210: E RECORD KEY data item extends beyond minimum record size for file: *file-name-1*

The data-name declared for a record key of the indicated file-name refers to a data item which extends outside the minimum record size for the file. All record keys must be totally contained within the minimum record size.

0211: E RECORD KEY data item is not defined in record associated with file: *file-name-1*

The data-name declared for a record key of the indicated file-name refers to a data item which is not defined in a record associated with the file-name. All record keys must be defined within a record associated with the file.

0212: E RECORD KEY data item has same offset as another record key for file: *file-name-1*

The data-name declared for a record key of the indicated file-name refers to a data item which has the same leftmost character offset as another record key of that file-name. No two keys may share the same leftmost character position.

0213: E RECORD KEY data item length exceeds 255 characters for file: *file-name-1*

The data-name declared for a record key of the indicated file-name refers to a data item with a length of more than 255 characters.

0214: E RECORD KEY data item must be alphanumeric or unsigned numeric DISPLAY item for file: *file-name-1*

The data-name declared as a record key of the indicated file-name refers to a data item which does not have an allowed data type. A record key data item must be category alphanumeric or an unsigned numeric data item with DISPLAY usage.

0215: E RECORD KEY data-name has error in its data description entry for file: *file-name-1*

The data-name declared for a record key of the indicated file-name refers to a data item which has an error in its description.

0216: E RECORD KEY operand must refer to data item for file: *file-name-1*

The data-name declared for a record key of the indicated file-name refers to a nondata item.

0217: E RECORD KEY data item must not be table element for file: *file-name-1*

The data-name declared for a record key of the indicated file-name refers to a data item which is described with the OCCURS clause or is subordinate to an item described with the OCCURS clause. Record keys may not be table items.

0218: E RECORD KEY data-name is not unique for file: *file-name-1*

The data-name declared for a record key of the indicated file-name refers to two or more data items; the qualification is ambiguous.

0219: E RECORD KEY data-name is not defined for file: *file-name-1*

The data-name declared for a record key of the indicated file-name is undefined.

0220: E RECORD DEPENDING data-name must be defined in Working-Storage or Linkage Section for file: *file-name-1*

The data-name used in the DEPENDING ON phrase of the RECORD IS VARYING clause has been defined in the wrong section of the Data Division. It must be defined in the Working-Storage Section or the Linkage Section.

0221: E RECORD DEPENDING data item must be able to contain maximum record size for file: *file-name-1*

The data-name used in the DEPENDING ON phrase of the RECORD IS VARYING clause has not been defined to be large enough to hold the number that represents the maximum number of characters needed for the record. The data item description should be changed so that it can contain the value of the maximum record size for the file.

0222: E RECORD DEPENDING data item must be unsigned integer for file: *file-name-1*

The data-name specified in the DEPENDING ON phrase of the RECORD IS VARYING clause must be defined as an elementary unsigned integer.

0223: E Record description sizes conflict with RECORD clause specification for file: *file-name-1*

The declaration of the file record size in the RECORD clause does not match the size described by the record description entry or entries given. This includes specification of the RECORD IS VARYING format when only fixed-length records are described.

This error also occurs for a file described with the EXTERNAL clause and without a RECORD clause when there are multiple record descriptions of differing lengths. COBOL requires that if the RECORD clause is not specified for an external file, all the record description entries associated with the file connector must be the same length. This rule is one of the rules for the RECORD clause, which begin on page 4-15.

0224: E RECORD DEPENDING data-name has error in its data description for file: *file-name-1*

There is an error in the data description of the data-name used in the DEPENDING ON phrase of the RECORD IS VARYING clause.

0225: E RECORD DEPENDING operand must refer to data item for file: *file-name-1*

The data-name specified in the DEPENDING ON phrase of the RECORD IS VARYING clause is not defined as an elementary numeric data item.

0226: E RECORD DEPENDING data item must not be table element for file: *file-name-1*

The data-name specified in the DEPENDING ON phrase of the RECORD IS VARYING clause cannot be defined with an OCCURS clause or be subordinate to an OCCURS clause.

0227: E RECORD DEPENDING data-name is not unique for file: *file-name-1*

The data-name specified in the DEPENDING ON phrase of the RECORD IS VARYING clause is defined more than once and is not adequately qualified.

0228: E RECORD DEPENDING data-name is not defined for file: *file-name-1*

The data-name specified in the DEPENDING ON phrase of the RECORD IS VARYING clause has not been defined.

0229: E RECORD DEPENDING data item is wrong linkage item for file: *file-name-1*

The data-name specified in the DEPENDING ON phrase of the RECORD IS VARYING clause for the indicated file-name is defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, nor is it defined subordinate to such a data-name. The data-name should be included as a USING parameter or defined outside the Linkage Section.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0230: E Record length exceeds 65280 characters $\left\{\begin{matrix} : \textit{ file-name-1} \\ . \end{matrix}\right\}$

The maximum file record size allowed is 65280 characters.

0231: E RELATIVE KEY data item must not be defined in record area for file: *file-name-1*

The data-name declared for the relative key of the indicated file-name refers to a data item defined in a record associated with file-name.

0232: E RELATIVE KEY data item must be unsigned integer for file: *file-name-1*

The data-name declared for the relative key of the indicated file-name refers to a data item which is not a numeric integer.

0233: E RELATIVE KEY data-name has error in its data description entry for file: *file-name-1*

The data-name declared for the relative key of the indicated file-name refers to a data item which has an error in its description.

0234: E RELATIVE KEY operand must refer to data item for file: *file-name-1*

The data-name declared for the relative key of the indicated file-name refers to a nondata item.

0235: E RELATIVE KEY data item must not be table element for file: *file-name-1*

The data-name declared for the relative key of the indicated file-name refers to a data item which is described with the OCCURS clause or is subordinate to an item described with the OCCURS clause.

0236: E RELATIVE KEY data-name is not unique for file: *file-name-1*

The data-name declared for the relative key of the indicated file-name refers to two or more data items; the qualification is ambiguous.

0237: E RELATIVE KEY data-name is not defined for file: *file-name-1*

The data-name declared for the relative key of the indicated file-name is undefined.

0238: E RELATIVE KEY data item is wrong linkage item or is not external item for file: *file-name-1*

The data-name declared for the relative key of the indicated file-name refers to a linkage data item which is not subordinate to an item in the Procedure Division header USING phrase.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0239: E FILE STATUS data item must have length of two characters for file: *file-name-1*

The data-name declared for the file status data item of the indicated file-name refers to a data item which is not two characters in length.

0240: E FILE STATUS data item must be alphanumeric for file: *file-name-1*

The data-name declared for the file status data item of the indicated file-name refers to a data item which is not of the category alphanumeric.

0241: E FILE STATUS data-name has error in its data description for file: *file-name-1*

The data-name declared for the file status data item of the indicated file-name refers to a data item which has an error in its description.

0242: E FILE STATUS operand must refer to data item for file: *file-name-1*

The data-name declared for the file status data item of the indicated file-name refers to a nondata item.

0243: E FILE STATUS data item must not be table element for file: *file-name-1*

The data-name declared for the file status data item of the indicated file-name refers to a data item which is described with the OCCURS clause or is subordinate to an item described with the OCCURS clause. The file status data item may not be a table item.

0244: E FILE STATUS data-name is not unique for file: *file-name-1*

The data-name declared for the file status data item of the indicated file-name refers to two or more data items; the qualification is ambiguous.

0245: E FILE STATUS data-name is not defined for file: *file-name-1*

The data-name declared for the file status data item of the indicated file-name refers to an undefined data item.

0246: E FILE STATUS data item is wrong linkage item for file: *file-name-1*

The data-name declared for the file status data item of the indicated file-name refers to a linkage data item which is not subordinate to an item in the Procedure Division header USING phrase.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0247: E TOP data item must be unsigned integer for file: *file-name-1*

The data-name specified in the LINES AT TOP phrase of the LINAGE clause for the indicated file does not refer to an elementary unsigned numeric data item.

0248: E TOP data-name has error in its data description entry for file: *file-name-1*

The data-name specified in the LINES AT TOP phrase of the LINAGE clause for the indicated file has an error in its data description.

0249: E TOP operand must refer to data item for file: *file-name-1*

The data-name specified in the LINES AT TOP phrase of the LINAGE clause for the indicated file does not refer to a valid data item described in the Data Division.

0250: E TOP data item must not be table element for file: *file-name-1*

The data-name specified in the LINES AT TOP phrase of the LINAGE clause for the indicated file cannot be defined with an OCCURS clause.

0251: E TOP data-name is not unique for file: *file-name-1*

The data-name specified in the LINES AT TOP phrase of the LINAGE clause for the indicated file is defined more than once and is not adequately qualified.

0252: E TOP data-name is not defined for file: *file-name-1*

The data-name specified in the LINES AT TOP phrase of the LINAGE clause for the indicated file has not been defined. An elementary unsigned numeric data entry in the Data Division is required.

0253: E TOP data item is wrong linkage item or is not external item for file: *file-name-1*

- The data-name specified in the LINES AT TOP phrase of the LINAGE clause for the indicated file has been defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, nor is it defined subordinate to such a data-name. The data-name should be included as a USING parameter or defined outside the Linkage Section.

  Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

- Or, the indicated file-name names an external file connector and the data-name specified in the LINES AT TOP phrase of the LINAGE clause does not possess the external attribute as required.

0254: E Level-number must be greater than level-number in previous data description entry which describes group.

The previous data description entry did not define an elementary data item, but the indicated level-number is less than or equal to the level-number of the previous data description entry.

0255: E Data-name in USING or GIVING phrase of Procedure Division header must not be described with REDEFINES clause.

The description of an operand that is specified in the USING or GIVING phrase in the Procedure Division header may not include a REDEFINES clause. The name of the original definition must be specified instead.

0256: E USAGE clause must not specify different usage than USAGE clause specified in containing group entry.

The USAGE clause indicated contradicts the USAGE clause for the group to which the subject item belongs.

0257: E VALUE clause must not be specified in data description entry when containing group entry has VALUE clause.

The VALUE clause indicated is given for a data item which belongs to a group for which a VALUE clause was also specified.

0258: E Hexadecimal literal has wrong character.

The indicated character within a hexadecimal literal is not a valid hexadecimal digit. The allowable characters are: 0 through 9, A through F, and a through f.

0259: E Identification Division has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the Identification Division as given in the source program.

0260: E Identifier has error in its data description entry.

The indicated identifier refers to an item which has an error in its description.

0261: E Identifier is not unique.

The identifier refers to two or more items; the qualification is ambiguous.

0262: E Identifier expected. Literal is not permitted here.

The context requires an identifier instead of a literal.

0263: E Identifier is not defined.

The identifier is undefined. This includes qualification errors such as incorrect qualifiers.

This error may also indicate that the identifier is defined outside of—but is not accessible to—the current program because it either does not have the global attribute or is not defined in a program which contains the current program.

0264: E PERFORM statement must not refer to procedure in different independent segment.

The procedure-name must not refer to a different independent segment than the independent segment containing the PERFORM statement.

0265: E Index data item is not permitted as conditional variable.

The associated conditional variable is an index data item.

0266: E Index data item is not permitted here.

The context does not allow an index data item.

0267: E Neither index-name nor index data item is permitted here.

The context does not allow index-names or index data items.

0268: E User-defined word previously defined for use that does not permit its use as index-name.

The index-name is already defined and cannot be redefined.

0269: E Index-name is not permitted here.

The context does not allow index-names.

0270: E Index-name may access another table only if both tables have same element size.

An index-name cannot be used with a table other than the one with which it is associated unless there is an exact match in the number of character positions in both tables.

0271: E Value of integer that specifies minimum is greater than value of integer that specifies maximum.

The second integer is less than the first integer in the pair of integers indicated.

0272: E Integer expected. Nonnumeric literal is not permitted here.

The context requires an integer numeric literal, but a nonnumeric literal was found.

0273: E Unsigned integer expected. Signed integer is not permitted here.

The context requires an unsigned integer, but a signed integer was found.

0274: E Integer value that exceeds 65535 is not permitted here.

The indicated integer has a value too large for the context in which it was used. The maximum integer value in such contexts is 65535.

0275: E Nonzero integer value expected.

The context requires a nonzero integer.

0276: E I-O-CONTROL paragraph has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the I-O-CONTROL paragraph as given in the source program.

0277: E Expected file-name that does not refer to sort-merge file.

The context requires a file-name which refers to an I-O file; that is, a file-name of a sequential, relative or indexed organization file not defined as a sort-merge file.

0278: E Expected record-name of file that is not sort-merge file.

The context requires a record-name associated with an I-O file; that is, a sequential, relative or indexed organization file record-name not defined as a sort-merge record. The record-name may be qualified by the file-name of the file with which it is associated, but may not be subscripted or reference modified.

0279: E JUSTIFIED clause is not permitted for any data item that is group, numeric, edited, index, or pointer data.

The JUSTIFIED clause is given in the data description entry in conflict with other data description clauses specified for the same subject.

0280: E Sort-merge key is not defined in record associated with sort-merge file.

The indicated data-name is not associated with the sort-merge file specified in this statement. That is, the data-name is not defined as a record of the file nor is it defined subordinate to a record of the file. Thus, the data-name does not refer to data that can be used as a key for the SORT or MERGE operation.

0281: E Sort-merge key must not be table element.

The indicated sort-merge key data-name is described with the OCCURS clause or is defined subordinate to an item described with the OCCURS clause. That is, the data-name refers to a table data item and thus cannot be used as a key for the SORT or MERGE operation.

0282: E Sort-merge key must not be variable size group.

The indicated sort-merge key data-name refers to a group which contains a data item described with Format 2 of the OCCURS clause. That is, the data-name refers to a variable length data item and thus cannot be used as a key for the SORT or MERGE operation.

0283: E Level-number must be 01-49, 66, 77, 78, or 88.

The compiler expected a valid level-number at the indicated place in the source, but did not find one of the valid level-number values, which are 01 through 49, 66, 77, 78, and 88.

0284: E Level-number 77 data description entry must describe elementary data item.

A level-number 77 data description entry did not describe an elementary data item. A PICTURE clause or a USAGE IS INDEX clause is required in level-number 77 data description entries.

0285: E Level-number 88 condition-name expected.

The indicated item is not a level-number 88 conditional variable condition-name. The format of the SET statement used requires a conditional variable condition-name.

0286: E ADVANCING mnemonic-name must not be specified when file-name is described with LINAGE clause.

The ADVANCING mnemonic-name phrase of the WRITE statement cannot be used for a file that is described with a LINAGE clause.

0287: E ADVANCING TO LINE and AT END-OF-PAGE phrases permitted only when file-name described with LINAGE clause.

The ADVANCING TO LINE and END-OF-PAGE phrases of the WRITE statement are allowed only when the file is described with the LINAGE clause.

0288: E Data-name must have level-number 01 or 77 data description entry in Linkage Section.

A data-name in the USING or GIVING phrase of the Procedure Division header does not refer to a data item described with level-number 01 or 77.

0289: E Data-name must be described in Linkage Section.

A data-name in the Procedure Division USING header phrase does not reference a data item defined in the Linkage Section of the program.

0290: E Level-number 01 or 77 expected in Linkage Section.

An entry in the Linkage Section of the Data Division is neither a record description entry (level-number 01) nor a 77 level description entry (level-number 77).

0291: E Literal expected. Identifier is not permitted here.

The context requires a literal.

0292: E MEMORY SIZE clause requires either WORDS, CHARACTERS, or MODULES option.

There is a syntax error in the MEMORY clause of the OBJECT-COMPUTER paragraph. A memory size option was incorrect or omitted. The allowable options are WORDS, CHARACTERS or MODULES.

0293: E Repeated file-name in MERGE statement is not permitted.

A file-name is repeated within a MERGE statement. File-names must not be repeated within the MERGE statement.

0294: E USING phrase of MERGE statement requires two or more file-names.

Two or more USING files are required for a MERGE statement, but only one is given.

0295: E User-defined word previously defined for use that does not permit its use as mnemonic-name.

The user-defined word is already defined and cannot be redefined as a mnemonic-name.

0296: E Alphanumeric edited or alphabetic data item must not be moved to numeric or numeric edited data item.

The MOVE statement is wrong because it attempts to move an alphanumeric edited or alphabetic data item to a numeric edited or numeric data item.

0297: E Noninteger numeric data item must not be moved to alphabetic, alphanumeric, or alphanumeric edited data item.

The MOVE statement is wrong because it attempts to move a noninteger numeric data item to a nonnumeric data item.

0298: E Numeric edited data item must not be moved to alphabetic data item.

The MOVE statement is wrong because it attempts to move a numeric edited data item to an alphabetic data item.

0299: E Numeric data item must not be moved to alphabetic data item.

The MOVE statement is wrong because it attempts to move a numeric data item to an alphabetic data item.

0300: E Zero length literal is not permitted. This may be caused by extraneous plus sign, minus sign, or period.

The indicated literal has zero length. For a numeric literal, this means no digit positions are defined. For a nonnumeric literal, this means that there are no characters enclosed in the quotation marks. This error may also result from the presence of an extraneous plus sign, minus sign or period in the source text.

0301: E Numeric literal exceeds 30 decimal digits.

The indicated numeric literal defines more than 30 digit positions.

0302: E Level-number must be greater than level-number in previous data description with OCCURS DEPENDING ON clause.

The indicated level-number is wrong because it is less than or equal to the level-number of an item described with the OCCURS . . . DEPENDING clause and is not the beginning of a new record description entry.

0303: E Data item having OCCURS DEPENDING ON clause must not be subordinate to data item having OCCURS clause.

The OCCURS . . . DEPENDING clause is specified subordinate to a data item described with the OCCURS clause.

0304: E OCCURS DEPENDING in redefinition is not permitted.

The OCCURS . . . DEPENDING clause is specified for a data item which is described with the REDEFINES clause or is subordinate to an item described with the REDEFINES clause.

0305: E More than 2046 external items are specified in separately compiled program, including any contained programs.

The implementation limit of 2046 external items in a single separately compiled program, including any of its contained programs, has been exceeded in the current program.

0306: E Pseudo-text-1 may not be empty. One or more text words are required here.

The left pseudo-text operand in a BY phrase must not be empty.

0307: E Open mode INPUT, OUTPUT, I-O, or EXTEND option expected.

A wrong open mode is specified. The allowed open mode options are EXTEND, INPUT, I-O and OUTPUT.

0308: E Combination of operands in SET statement is wrong.

The combination of operands specified is wrong because of their data types and the context. (See Table 6-10 on page 6-181 for valid combinations.)

0309: E Positive integer expected. Negative integer is not permitted here.

The indicated integer cannot be negative.

0310: E Paragraph-name and section-name must not be same user-defined word.

A paragraph and a section must not be given the same name.

0311: E Cannot refer to paragraph-name that is not unique within section.

The specified paragraph is defined more than once within the specified section.

0312: E PERFORM procedure-names must be in same declarative section: *procedure-name-1*

A PERFORM statement is wrong because either the entry or exit procedure-name refers to a procedure in a declaratives section and the other refers to a procedure not in the same declaratives section.

0313: E PERFORM entry procedure-name must not be in different independent segment: *procedure-name-1*

A PERFORM statement in an independent segment is wrong because the entry procedure-name refers to a procedure in a different independent segment.

0314: E PERFORM entry procedure-name is not unique: *procedure-name-1*

A PERFORM statement is wrong because the entry procedure-name is ambiguous. Qualification is required to yield a unique reference.

0315: E PERFORM entry procedure-name is not defined: *procedure-name-1*

A PERFORM statement is wrong because the entry procedure-name is undefined. This includes qualification errors such as a qualified section-name.

0316: E PERFORM exit procedure-name must be in same independent segment as entry: *procedure-name-1*

A PERFORM statement is wrong because its exit procedure-name refers to a procedure in a different independent segment than the segment containing the entry procedure.

0317: E PERFORM exit procedure-name is not unique: *procedure-name-1*

A PERFORM statement is wrong because the exit procedure-name is ambiguous. Qualification is required to yield a unique reference.

0318: E PERFORM exit procedure-name is not defined: *procedure-name-1*

A PERFORM statement is wrong because the exit procedure-name is undefined. This includes qualification errors such as a qualified section-name.

0319: E Period space separator expected.

The context requires a period space separator at the indicated point in the source program.

0320: E BLANK WHEN ZERO clause must not be specified with PICTURE character-string containing symbols '*' or 'S'.

The BLANK WHEN ZERO clause is used to describe a data item which specifies asterisk zero suppression or an operational sign in its PICTURE character-string.

0321: E Wrong character in PICTURE character-string.

The indicated character in the PICTURE character-string is not a valid PICTURE character.

0322: E PICTURE clause must not be specified for index or pointer data item.

The PICTURE clause has been used to describe an index (USAGE IS INDEX) or pointer (USAGE IS POINTER) data item. These types of data items do not allow a PICTURE clause.

0323: E Letter 'R' is missing from 'CR' symbol in PICTURE character-string.

The PICTURE character-string contains a “C” not followed by “R”.

0324: E PICTURE character-string describes data item with length that exceeds 65280 characters.

The number of character positions described by the PICTURE character-string for a single data item cannot exceed 65280 character positions.

0325: E Letter 'B' is missing from 'DB' symbol in PICTURE character-string.

The PICTURE character-string contains a “D” not followed by “B”.

0326: E Ending pseudo-text delimiter is missing.

The nearest preceding pseudo-text delimiter was an opening pseudo-text delimiter for which no closing pseudo-text delimiter has been found.

0327: E Repeated character in first literal of INSPECT CONVERTING statement is not permitted.

A character must not appear more than once in a CONVERTING literal.

0328: E Fixed insertion currency must be first symbol in PICTURE character-string.

The PICTURE character-string contains a fixed insertion currency symbol which is not the leftmost character in the character-string, except for either a ‘+’ or a ‘–’ symbol.

0329: E Fixed insertion sign must be first or last symbol in PICTURE character-string.

The PICTURE character-string contains a fixed insertion sign character which is not the leftmost or rightmost character in the character-string.

0330: E PICTURE character-string has wrong combination of symbols '.', 'P', and 'V'.

The PICTURE character-string contains combinations of scaling characters (P, decimal-point, V) such that the decimal point position is defined in more than one place.

0331: E Symbol in PICTURE character-string is wrong for nonnumeric data item.

The PICTURE character-string was nonnumeric up to the indicated character which is not permitted in a nonnumeric PICTURE character-string.

0332: E PICTURE character-string for numeric or numeric edited data item exceeds 30 decimal digits.

The PICTURE character-string defines a numeric or numeric edited data item with more than 30 digit positions.

0333: E PICTURE character-string symbol combination wrong for PICTURE precedence rules.

The indicated character in a numeric or numeric edited PICTURE character-string violates the precedence rules. (See Table 4-4 on page 4-45.)

0334: E PICTURE character-string for numeric or numeric edited data item must include digit positions.

The PICTURE character-string defines a numeric or numeric edited data item without any digit positions.

0335: E PICTURE character-string must not have digit positions both to left and right of symbols 'P'.

The PICTURE character-string defines digit positions both to the left and right of P characters.

0336: E Symbols 'CR', 'DB', 'S', 'V', and '.' must occur only once in PICTURE character-string.

The indicated character in the PICTURE character-string is repeated when it must occur as a single character.

0337: E Symbol is not permitted in PICTURE character-string for signed numeric item.

The indicated character in the PICTURE character-string is not allowed in a signed numeric data item (that is, a character-string starting with S).

0338: E PICTURE character-string and USAGE clause are not compatible.

The PICTURE character-string describes a data item which conflicts with the USAGE declared for the data item (for example, nonnumeric picture with COMP usage).

0339: E Section-name header required because declaratives specified in same program.

The nondeclarative portion of the Procedure Division must be sectioned when declaratives are defined.

0340: E Procedure-name is not unique: *procedure-name-1*

The indicated procedure reference is ambiguous and requires qualification to yield a unique procedure reference.

0341: E Procedure-name required in GO TO statement that is not alterable.

The indicated Format 1 GO TO statement does not occur in a single statement paragraph (alterable paragraph) and, therefore, must be followed by a procedure-name. This error may indicate that the procedure-name specified is a reserved word.

0342: E Procedure-name is not defined: *procedure-name-1*

The indicated procedure reference is not defined. This includes qualification errors such as a qualified section-name.

0343: E Alphabet-name specified for PROGRAM COLLATING SEQUENCE must not have duplicate character.

The alphabet-name specified in the PROGRAM COLLATING SEQUENCE clause refers to an alphabet defined with a duplicate character. Since a character can only have one collating position, a character must not be repeated in the definition of an alphabet-name specified as a collating sequence. Refer to the "ASCII Position" and "U.S. Character" columns in Appendix J, *Code-Set Translation Tables*, of the *RM/COBOL User's Guide*, for the exact correlation of ordinal position to native character. Informational messages 8 and 9 (see page B-2) are generated at the end of the program listing to provide the alphabet-name and duplicated character.

0344: E Data defined in program has combined length that exceeds 4 gigabytes.

The program defined data in excess of four gigabytes.

0345: E Object instructions generated by procedural statements in single segment exceeds 16 megabytes.

The program required object instructions—generated for Procedure Division statements—that were in excess of 16 megabytes for a single segment. Approximately 10 bytes are generated per source statement.

0346: E Program-name in END PROGRAM header must equal program-name in PROGRAM-ID paragraph.

The program-name specified in the END PROGRAM header does not match the program-name specified in the PROGRAM-ID paragraph of the Identification Division.

0347: E Statement does not permit data item described with symbol 'P' in its PICTURE character-string.

Use of a data item described with the scaling position character P in its PICTURE character-string is not allowed in the context of the indicated statement in the source program.

0348: E Qualification of section-name is not permitted.

A section-name is qualified in the source program. Section-names must be unique in the set of procedure-names for a given source program and cannot be qualified.

0349: E File-name described with ACCESS MODE RANDOM clause is not permitted here.

The context does not allow a file defined with RANDOM access mode. The indicated file-name must be described with SEQUENTIAL or DYNAMIC access mode.

0350: E ASSIGN clause for file: must specify RANDOM, DISK, or DISC.

The context requires a file assigned to a mass-storage device-name, which is RANDOM, DISK or DISC.

0351: E KEY phrase is permitted only for random or dynamic access indexed file without NEXT or PREVIOUS phrases.

The KEY phrase is not allowed in the indicated READ statement because either *file-name-1* does not refer to an indexed organization file, *file-name-1* refers to a file that has sequential access mode, or the NEXT or PREVIOUS phrase is specified in the same READ statement.

0352: E Record description entry must begin with level-number 01.

The level-number of a record entry is not 01.

0353: E File description entry must be followed by one or more record description entries.

The context requires a record entry description at the indicated point in the source program.

0354: E Number of record keys or record key segments exceeds 255.

More than 255 record keys are defined for a file or more than 255 record key segments are defined for a file. RM/COBOL allows at most 255 record keys or record key segments.

0355: E RECORD KEY and ALTERNATE RECORD KEY clauses permitted only in indexed file control entry.

The RECORD KEY clause is given for a file which is not indexed organization.

0356: E Identifier must refer to record key or to data item aligned on record key associated with file-name.

Context requires a data-name which is a record key of the associated file-name or, for a START statement, a data-name that refers to a data item whose leftmost character position is the same as the leftmost character position of a record key of the associated file-name.

0357: E RECORD KEY clause is required in indexed file control entry.

An indexed organization file must be described with the RECORD KEY clause. A prime record key is required for indexed files.

0358: E REDEFINES clause not permitted for level-number 01 entries in this section.

The REDEFINES clause may not be used in level-number 01 entries of the File Section or Communication Section. When multiple level-number 01 entries are subordinate to a file description entry (FD) or communication-description-entry (CD) in these sections, all but the first entry implicitly redefine the first entry.

0359: E REDEFINES cannot specify this data-name.

The data-name specified in a REDEFINES clause is wrong because it is not that of the last allocated data item nor the data-name of the last redefinition at the same level.

0360: E REDEFINES cannot specify data-name described with OCCURS clause.

The data item to be redefined cannot be described with an OCCURS clause.

0361: E Size of redefinition exceeds size of item referred to by REDEFINES clause.

The indicated data-name defines an area of storage larger than the area it is redefining and is not described with level-number 01.

0362: E VALUE clause is not permitted in redefinition.

The VALUE clause is used to describe a data item which is also described with the REDEFINES clause or which is subordinate to a data item described with the REDEFINES clause.

0363: E REDEFINES is not permitted for group containing variable occurrence data item.

A data item that is variable length as a result of Format 2 of the OCCURS clause cannot be redefined.

0364: E Nonnumeric to numeric relation requires the numeric object to be DISPLAY usage.

The indicated relation condition compares a nonnumeric value to a numeric value, but the numeric object is not DISPLAY usage as required for such a relation condition.

0365: E Nonnumeric to numeric relation requires the numeric object to be an integer.

The indicated relation condition compares a nonnumeric value to a numeric value, but the numeric object is not an integer as required for such a relation condition.

0366: E Relational operator is not permitted in START statement.

The context does not allow the indicated relation.

0367: E Data item, literal, or arithmetic expression expected for object of relational operator.

The conditional expression is syntactically incorrect because a conditional expression was found where an arithmetic expression, nonnumeric data item or nonnumeric literal was required as the object of a preceding relation operator.

0368: E Relational operator expected.

The context requires a relation operator.

0369: E Relational operator specified without subject data item, literal, or arithmetic expression.

The conditional expression is syntactically incorrect because a relation condition with no subject was specified and the relation is not a valid abbreviated relation condition.

0370: E RELATIVE KEY phrase permitted only in relative file control entry.

The RELATIVE KEY phrase is specified for a file which does not have relative organization. The RELATIVE KEY phrase is not allowed in an indexed or sequential file control entry.

0371: E RELATIVE KEY phrase required for relative file referred to in START statement or with random or dynamic access.

A relative organization file with random or dynamic access must be described with the RELATIVE KEY phrase. Also, if a START statement refers to a relative file, the RELATIVE KEY phrase must be specified for that file.

0372: E RENAMES clause must not refer to data-name described with level-number 01, 66, or 77.

The object data-name of a RENAMES clause is wrong because it is described with level-number 01, 66 or 77.

0373: E RENAMES clause must not refer to data-name described with OCCURS clause.

The object data-name of a RENAMES clause is wrong because it is described with the OCCURS clause or is subordinate to a data item described with the OCCURS clause.

0374: E Second data item in THRU phrase of RENAMES clause must not begin to left of first data item in that phrase.

The beginning of the area described by *data-name-3* begins to the left of the area described by *data-name-2* in a RENAMES *data-name-2* THRU *data-name-3* clause.

0375: E Second data item in THRU phrase of RENAMES clause must end to right of first data item in that phrase.

The end of the area described by *data-name-3* is not to the right of the area described by *data-name-2* in a RENAMES *data-name-2* THRU *data-name-3* clause.

0376: E RENAMES of group containing variable occurrence data item is not permitted.

The object data-name of a RENAMES clause is described such that it is variable length as defined in the OCCURS clause.

0377: E Repeated RERUN clause for file-name is not permitted.

A RERUN statement has been repeated for the same file.

0378: E ON phrase required for this format of RERUN clause.

An ON phrase is needed in the RERUN clause when either an END OF REEL or END OF UNIT phrase is used and the file-name associated with the END OF REEL or END OF UNIT is not an output file, or when the condition-name format of the RERUN clause is used.

0379: E Rerun-name required in ON phrase for this format of RERUN clause.

The ON phrase with the rerun-name option must be specified if either the RECORDS or CLOCK-UNITS phrase is used.

0380: E RERUN clause has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the RERUN clause of the I-O-CONTROL paragraph as given in the source program.

0381: E RESERVE clause integer value exceeds 255.

The integer given in the RESERVE AREAS clause specifies that more than 255 input-output areas be reserved.

0382: E Computer-name must be user-defined word instead of reserved word.

The indicated computer-name is a reserved word; a user-defined word must be given for the computer-name.

0383: E Text-name must not be reserved word. Nonnumeric literal may be used to avoid this conflict.

The indicated text-name is a reserved word; a user-defined word must be given for the text-name.

0384: E User-defined word expected instead of reserved word.

Context requires a user-defined word at the indicated position in the source program, but a reserved word was found.

0385: E Right parenthesis missing in PICTURE character-string.

The PICTURE character-string contains a repeat count which is not properly terminated with a right parenthesis. The right parenthesis is missing, possibly because it is within the identification area (columns 73 through 80) of the source record or because text follows the integer specifying the count.

0386: E File-name may not be specified more than once in SAME AREA clause.

The indicated file-name is specified more than once in a SAME AREA clause.

0387: E All file-names in SAME AREA clause must also occur in any associated SAME RECORD AREA clause: *file-name-1*

The indicated file-name is specified in a SAME AREA clause with another file-name which is also specified in a SAME RECORD AREA clause. The indicated file-name is not specified in the SAME RECORD AREA clause as required.

0388: E All file-names in SAME AREA clause must also occur in any associated SAME SORT/SORT-MERGE AREA clause: *file-name-1*

The indicated file-name is specified in a SAME AREA clause with another file-name which is also specified in a SAME SORT AREA clause. The indicated file-name is not specified in the SAME SORT AREA clause as required.

0389: E Sort-merge file-name must not be specified in SAME AREA clause.

The indicated file-name refers to a sort-merge file and is, therefore, not allowed in a SAME AREA clause.

0390: E File-name must not be specified more than once in SAME RECORD AREA clause.

The indicated file-name is specified more than once in a SAME RECORD AREA clause.

0391: E Repeated sort-merge file-name in SAME SORT AREA clause is not permitted.

The indicated file-name is a sort-merge file that is specified more than once in a SAME SORT AREA clause or SAME SORT-MERGE AREA clause.

0392: E At least one file-name in SAME SORT AREA clause must be sort-merge file-name.

The indicated SAME SORT AREA clause or SAME SORT-MERGE AREA clause does not contain a sort-merge file-name as required.

0393: E Sort-merge file description entry (SD entry) has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the sort-merge file description entry as given in the source program.

0394: E Section header is not permitted in Procedure Division that begins with paragraph header.

A section definition is not allowed here because the Procedure Division did not begin with a section. The section definition is accepted.

0395: E Section-name is not unique.

The indicated user-defined word is already defined as a section-name or a paragraph-name and, therefore, cannot be defined as a new section-name.

0396: E Section-name expected.

Context requires a section-name. A paragraph-name may not be used as a qualifier.

0397: E Repeated SEGMENT-LIMIT clause is not permitted.

The SEGMENT-LIMIT clause has been defined more than once.

0398: E Segment-number specified in SEGMENT-LIMIT clause must be 01-49.

The segment-number in the SEGMENT-LIMIT clause must be within the range of 1 through 49.

0399: E Segment-number exceeds 127.

The indicated segment-number is larger than the limit of 127. The last valid segment-number is used instead.

0400: E Random or dynamic access is not permitted for sequential organization file or for EXTEND open mode.

The context requires a sequential access file. A sequential organization file must be described implicitly or explicitly as having sequential access. The EXTEND open mode may only be specified for files described implicitly or explicitly as having sequential access.

0401: E File-name of sequential organization file is not permitted here.

The context does not allow a sequential organization file.

0402: E File-name of sequential organization file expected.

The context requires a sequential organization file.

0403: E SIGN clause not permitted with unsigned PICTURE character-string or usage other than DISPLAY.

The SIGN clause is given in conflict with other data description entries.

0404: E LEADING or TRAILING option expected in SIGN clause.

The SIGN clause specifies an unrecognized option. The valid options are LEADING and TRAILING.

0405: E Sort-merge file-name expected.

The context requires a file-name that refers to a sort-merge file.

0406: E Record-name associated with sort-merge file expected.

The context requires a record-name associated with a sort-merge file. The record-name may be qualified by the file-name of the sort-merge file with which it is associated, but may not be subscripted or reference modified.

0407: E Sort-merge file must have sequential organization.

The file-name following an SD level-indicator must reference a sequential organization file.

0408: E Clauses specified in wrong order within SPECIAL-NAMES paragraph.

The clauses in the SPECIAL-NAMES paragraph are not listed in the order shown in the paragraph skeleton. The required order is mnemonic-names, ALPHABET, SYMBOLIC CHARACTERS, CLASS, CURRENCY SIGN and DECIMAL-POINT. Clauses not needed may be omitted.

0409: E SPECIAL-NAMES paragraph has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the SPECIAL-NAMES paragraph as given in the source program.

0410: E Alphabet-name, class-name, or mnemonic-name is not permitted here.

The context does not allow a special-name such as a mnemonic-name or alphabet-name as given.

0411: E Data-name must not refer to data item that is longer than associated record key.

The data-name given in the START statement relation for an indexed organization file does not reference a data item which is subordinate to its associated record key.

0412: E Data-name must refer to data item that is alphanumeric or unsigned numeric with DISPLAY usage.

The data-name given in the START statement relation for an indexed organization file does not reference a data item with an allowed data type. The data item must be described as category alphanumeric or as an unsigned numeric data item with DISPLAY usage.

0413: E Data-name must refer to relative key data item.

The data-name given in the START statement relation for a relative organization file is not the relative key data-name as required.

0414: E Statement has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the Procedure Division verbs as given in the source program.

0415: E Identifier may not refer to table element.

The context does not allow a subscripted reference. The data item specified here must not be described with the OCCURS clause or be subordinate to an item described with the OCCURS clause.

0416: E Data item used as subscript must not be table element.

A data-name specified as a subscript is described with the OCCURS clause or is subordinate to a data item described with the OCCURS clause.

0417: E Too many subscripts for table element or missing right parenthesis.

The syntax of the subscripting for the identifier is incorrect. Either too many subscripts are specified or the right parenthesis is missing, possibly because it is in the identification area (columns 73–80) of the source record.

0418: E Identifier refers to table element and thus must be subscripted.

The indicated data-name must be subscripted to provide a unique reference.

0419: E Identifier needs more subscripts to form unique reference.

Too few subscripts were specified for the identifier. The data-name portion of the identifier refers to a table element which, in order to specify a unique reference, requires more subscripts than were specified.

0420: E Literal subscript value exceeds number of table elements.

The indicated literal subscript is greater than the maximum number of table elements that are defined in the OCCURS clause for the specified table. A relative subscript literal is limited to one less than the number of elements.

0421: E Switch-status condition-name expected.

The indicated context requires a switch condition-name, but a user word for some other entity was specified, such as a data-name, file-name or alphabet-name.

0422: E Switch-status condition-name is not defined.

The indicated context requires a switch condition-name, but an undefined user word was specified.

0423: E Mnemonic-name associated with external switch expected.

The indicated mnemonic-name is not associated with an external switch as required by the context. SWITCH-1 through SWITCH-8 or the synonyms UPSI-0 through UPSI-7 may be specified in the SPECIAL-NAMES paragraph to associate a mnemonic-name with an external switch.

0424: E Repeated ON STATUS or OFF STATUS phrase in switch-name clause is not permitted.

Two ON STATUS or OFF STATUS condition-names are defined in the same SPECIAL-NAMES clause for a switch implementor-name. The language syntax requires an ON/OFF or OFF/ON pair or a single ON or OFF status declaration.

0425: E ON STATUS or OFF STATUS phrase expected.

At least one ON or OFF STATUS condition-name must be associated with a switch-name.

0426: E User-defined word must begin with letter or digit.

A user-defined word must begin with a letter or digit.

0427: E User-defined word must contain at least one letter.

A data-name must contain at least one letter character.

0428: E Integer in SYMBOLIC CHARACTERS clause exceeds 256 or character code set size.

The integer specified in the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph represents the ordinal position of the character in the native character set or of the character set specified by alphabet-name in the IN phrase. Valid integer values for the native character set are 1 through 256. Valid integer values for an alphabet-name are dependent on the number of characters included in the description.

0429: E Symbolic-character already declared.

The user-defined word in the indicated SYMBOLIC CHARACTERS clause has been previously defined and cannot be used again.

0430: E User-defined word following ALL must be symbolic-character.

A symbolic-character name is required following the indicated figurative constant ALL.

0431: E Symbolic-character is not defined by SYMBOLIC CHARACTERS clause.

The indicated name following the figurative constant ALL is presumed to be a user-defined symbolic name but is not defined in a SYMBOLIC CHARACTERS clause.

0432: E SYNCHRONIZED clause is not permitted for group.

The SYNCHRONIZED clause was specified in conflict with other data description clauses specified in the same entry.

0433: E Identifier must not be subscripted by first index-name associated with table being searched.

The indicated identifier is subscripted by the first index-name of the table being searched by this SEARCH statement. In this context (for example, the VARYING phrase), such subscripting is disallowed.

0434: E OCCURS DEPENDING ON data-name must not be defined within table for table: *table-name-1*

The data-name for the DEPENDING ON phrase of the OCCURS clause cannot be in the variable-length portion of the table. This may occur with implicit redefinition of the table item.

0435: E OCCURS DEPENDING ON data item must be numeric integer for table: *table-name-1*

The data-name specified in the DEPENDING ON phrase of the OCCURS clause for the indicated table does not refer to a data item described as a numeric integer.

0436: E OCCURS DEPENDING ON data-name has error in its data description entry for table: *table-name-1*

The data-name specified in the DEPENDING ON phrase of the OCCURS clause for the indicated table refers to a data item that has an error in its description.

0437: E OCCURS DEPENDING ON must refer to data item for table: *table-name-1*

The data-name specified in the DEPENDING ON phrase of the OCCURS clause for the indicated table refers to a nondata item.

0438: E OCCURS DEPENDING ON data item must not be table element for table: *table-name-1*

The data-name specified in the DEPENDING ON phrase of the OCCURS clause for the indicated table refers to a data item described with the OCCURS clause or which is subordinate to a data item described with the OCCURS clause.

0439: E OCCURS DEPENDING ON data-name is not unique for table: *table-name-1*

The data-name specified in the DEPENDING ON phrase of the OCCURS clause for the indicated table refers to two or more data items; the qualification is ambiguous.

0440: E OCCURS DEPENDING ON data-name is not defined for table: *table-name-1*

The data-name specified in the DEPENDING ON phrase of the OCCURS clause for the indicated table is undefined.

0441: E OCCURS DEPENDING ON data item is wrong linkage item for table: *table-name-1*

The data-name for the DEPENDING ON phrase of the OCCURS clause is defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase nor is it defined subordinate to such a data-name. The data-name should be included as a USING parameter or defined outside the Linkage Section.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0442: E Table element length exceeds 65280 characters.

The maximum table element size has been exceeded. Up to 65280 characters may be defined.

0443: E More AND phrases than KEY phrases for table being searched is not permitted.

Too many AND phrases in the SEARCH ALL statement have been specified for the number of keys declared for the specified table.

0444: E OCCURS KEY data item must be defined in table for table: *table-name-1*

The indicated table key data-name is not associated with the data entry containing the OCCURS clause or is not subordinate to the entry containing the OCCURS clause.

0445: E Wrong key specification in SEARCH ALL statement.

The data items to be compared in the SEARCH ALL statement are not given in the same order as they appear in the OCCURS clause of the specified table, or an item which is not a key of the table has been listed.

0446: E OCCURS KEY data-name has error in its data description entry for table: *table-name-1*

The data-name in the KEY IS phrase of the OCCURS clause has an error in its data description.

0447: E OCCURS KEY operand must refer to data item for table: *table-name-1*

The data-name specified in the KEY IS phrase of the OCCURS clause is not a valid data item described in the Data Division.

0448: E OCCURS KEY data item must be same dimension as table for table: *table-name-1*

The data-name specified in the KEY IS phrase of the OCCURS clause is defined such that it requires a different number of subscripts than the table defined by the OCCURS clause.

0449: E OCCURS KEY data-name is not unique for table: *table-name-1*

The data-name specified in the KEY IS phrase of the OCCURS clause is defined more than once and is not adequately qualified.

0450: E OCCURS KEY data-name is not defined for table: *table-name-1*

The data-name specified in the KEY IS phrase of the OCCURS clause has not been defined.

0451: E Identifier must refer to data item with OCCURS clause in its data description entry.

The indicated identifier does not refer to a table, that is, a data item described with the OCCURS clause in its data description entry. For the SEARCH and SEARCH ALL statements, the data item to be searched must be a table. For the COUNT, COUNT-MAX, and COUNT-MIN special registers, the operand must be a table.

0452: E Identifier must refer to data item with INDEXED BY phrase in its data description entry.

The table specified in the SEARCH or SEARCH ALL statement does not contain an INDEXED BY phrase in the OCCURS clause as required.

0453: E Identifier must refer to data item with KEY phrase in its data description entry.

The table specified in the SEARCH ALL statement does not have a KEY IS phrase as required.

0454: E Library-name must not be reserved word. Nonnumeric literal may be used to avoid this conflict.

In the COPY statement, a reserved word was used to specify the library-name.

0455: E Library-name contains wrong character.

A wrong character was found in the library-name of the COPY statement.

0456: E Text-name contains wrong character.

The text-name in a COPY statement contains a wrong character.

0457: E Data item or literal must be unsigned integer.

The indicated literal must be specified as an unsigned integer or the indicated identifier must refer to a data item described as an unsigned integer.

0458: E USAGE clause has wrong format.

An unrecognized usage option is specified. The valid usage options are BINARY, COMP, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-3, COMPUTATIONAL-4, COMPUTATIONAL-6, COMP-1, COMP-3, COMP-4, COMP-6, DISPLAY, INDEX, PACKED-DECIMAL, and POINTER.

0459: E USING phrase in Procedure Division header exceeds 255 data-names.

The maximum number of operands in the USING phrase of the Procedure Division header has been exceeded. No more than 255 data-names may be specified.

0460: E Repeated data-name in USING phrase of Procedure Division header is not permitted.

A data-name is specified more than once in the USING phrase of the Procedure Division header.

0461: E VALUE clause is not permitted for index data item.

The VALUE clause is specified in conflict with other data description clauses specified in the same entry.

0462: E Numeric literal in VALUE clause must have value within range indicated by PICTURE clause.

The numeric literal specified in the VALUE clause for a numeric data item is incorrect for initialization of the data item as described by its PICTURE character-string because truncation of nonzero high-order digits was required.

0463: E Nonnumeric value in VALUE clause must not exceed size indicated by PICTURE clause.

The nonnumeric literal specified in the VALUE clause for a nonnumeric data item contains too many characters for initialization of the data item. Characters were truncated from the low-order (rightmost) end of the literal value.

0464: E Fixed size portion of variable size group exceeds 65280 characters.

The fixed-size portion of a group that has a variable-length table subordinate to it cannot be defined to be larger than 65280 character positions.

0465: E Verb expected.

The context requires a verb at the indicated position in the source program.

0466: E Level-number 01 or 77 expected in Working-Storage Section.

An entry in the Working-Storage Section of the Data Division is neither a record description entry (level-number 01) nor a 77 level description entry (level-number 77).

0467: E Identifier or condition-name must be subscripted by first index-name of table being searched.

The indicated reference to a table key data-name or condition-name is not subscripted with the first or only index-name of the table specified in the SEARCH ALL statement. Since only the first index-name of the table will be varied by the execution of the SEARCH ALL statement, the desired results cannot be obtained unless the subscripting is changed to include the first index-name of the table.

0468: E Paragraph has wrong format.

The indicated word, literal, character-string, or separator is incorrect syntax within the context of the paragraph as given in the source program.

0469: E Scope terminator does not match preceding unterminated verb.

The indicated scope terminator does not match a previously unmatched verb. For example, an ELSE is specified which is not paired with a previously unpaired IF. This error frequently occurs as a result of previous errors which caused the verb with which the scope terminator was meant to be paired to be either ignored by the compiler or already implicitly terminated by another scope terminator.

0470: E Data-name required in level-number 01 data description entry with GLOBAL or EXTERNAL clause.

The indicated data description entry must include a data-name since it is a record of a file described with either the GLOBAL or EXTERNAL clauses. FILLER or omission of the data-name is not allowed in level-number 01 record description entries for these files.

0471: E Identifier in SEARCH statement must be neither subscripted nor reference modified.

The indicated subscripting or reference modification is prohibited in the context in which it occurs. Although the item may be a table element and normally requires subscripting, the subscripting is prohibited in this context.

0472: E Paragraph generates object code that exceeds 32512 bytes.

The indicated Procedure Division paragraph has caused the generation of more than 32512 bytes of object code. The paragraph must be divided into two or more paragraphs by insertion of paragraph-names in the source program.

0473: E Sentence generates object code that exceeds 32512 bytes.

The indicated sentence has caused the generation of more than 32512 bytes of object code. The sentence must be divided into two or more sentences by insertion of the period space separator or by replacing a portion of the sentence with a PERFORM statement which refers to a paragraph or section containing the replaced statements.

0474: E DELIMITER IN and COUNT IN phrases require DELIMITED BY phrase in UNSTRING statement.

The DELIMITER IN and COUNT IN phrases are not allowed when the DELIMITED BY phrase is not specified in an UNSTRING statement.

0475: E Mnemonic-name must be associated with low-volume-I-O-name in SPECIAL-NAMES paragraph.

The indicated user-defined word is not a mnemonic-name defined in the SPECIAL-NAMES paragraph as being associated with a low-volume-I-O-name (for example, CONSOLE). The context requires a mnemonic-name associated with a low-volume-I-O-name: the mnemonic-name in the indicated context may not be associated with a switch-name or with a feature-name.

0476: E Identifier in INTO phrase of STRING statement must not refer to edited data item.

The indicated identifier refers to an edited data item in a context which does not allow edited data items.

0477: E Two or more file-names in MERGE statement must not be specified in one MULTIPLE FILE TAPE clause.

Two or more files specified in a MERGE statement are listed in the same MULTIPLE FILE TAPE clause in the I-O-CONTROL paragraph.

0478: E Two or more file-names in MERGE statement must not be specified in one SAME AREA or SAME RECORD AREA clause.

Two or more files specified in a MERGE statement are listed in the same SAME AREA or SAME RECORD AREA clause in the I-O-CONTROL paragraph.

0479: E Sort-merge key data item extends beyond minimum record size for sort-merge file.

A data-name specified in a KEY phrase of a SORT or MERGE statement refers to a data item which is not totally contained within the minimum record length of the sort-merge file.

0480: E Minimum record length conflicts with variable length sort-merge file or GIVING file.

The minimum record length of a USING file is less than the minimum record length of a sort-merge file with variable-length records or the minimum record length of the sort-merge file is less than the minimum record length of a GIVING file with variable-length records.

0481: E PADDING CHARACTER clause only permitted in sequential file control entry.

The PADDING CHARACTER clause may be specified only for sequential files.

0482: E RECORD DELIMITER clause only permitted in sequential file control entry.

The RECORD DELIMITER clause may be specified only for sequential files.

0483: E Conditional operator expected.

The context suggests that a class-name is intended at the indicated position, but the specified identifier is not a class-name.

0484: E CLASS clause has error in definition of class-name.

There is an error in the declaration of the specified class-name.

0485: E Cd-name of I-O CD entry expected.

An I-O cd-name must be specified in the context of the statement as given in the source program. An I-O cd-name is required with the DISABLE I-O and ENABLE I-O statements.

0486: E PADDING CHARACTER data item must be alphanumeric for file: *file-name-1*

The data-name declared in the PADDING CHARACTER clause for the indicated file-name must refer to a data item of category alphanumeric.

0487: E PADDING CHARACTER data-name has error in its data description entry for file: *file-name-1*

The data-name declared in the PADDING CHARACTER clause for the indicated file-name refers to a data item that has an error in its description.

0488: E PADDING CHARACTER operand must refer to data item for file: *file-name-1*

The data-name declared in the PADDING CHARACTER clause for the indicated file-name refers to a nondata item.

0489: E PADDING CHARACTER data item must not be table element for file: *file-name-1*

The data-name declared in the PADDING CHARACTER clause for the indicated file-name refers to a data item that is described with the OCCURS clause or is subordinate to an item described with the OCCURS clause. The padding character data item may not be a table item.

0490: E PADDING CHARACTER data-name is not unique for file: *file-name-1*

The data-name declared in the PADDING CHARACTER clause for the indicated file-name refers to two or more data items; the qualification is ambiguous.

0491: E PADDING CHARACTER data-name is not defined for file: *file-name-1*

The data-name declared in the PADDING CHARACTER clause for the indicated file-name is undefined.

0492: E PADDING CHARACTER data item is wrong linkage item or is not external item for file: *file-name-1*

The data-name declared in the PADDING CHARACTER clause for the indicated file-name refers to a linkage data item which is not subordinate to an item in the Procedure Division header USING phrase.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0493: E Selection object is incompatible with corresponding selection subject.

When a selection object is specified by a condition or by the words TRUE or FALSE, the corresponding selection subject must also be a condition or either of the words TRUE or FALSE; it may not be an identifier, a literal or an arithmetic expression. When a selection object is an identifier, literal or arithmetic expression, the corresponding selection subject must also be an identifier, a literal or an arithmetic expression.

0494: E Operands of THROUGH phrase must have same class for selection subject or object.

The two operands connected by a THROUGH phrase must be of the same class: numeric, alphanumeric or alphabetic.

0495: E REPLACING phrase of INITIALIZE statement must not repeat any given category.

The REPLACING phrase of the INITIALIZE statement specifies the same category in different BY phrases. Any given category must not be specified more than once in the REPLACING phrase.

0496: E Integer expected. Identifier is not permitted here.

The context requires an integer numeric literal, but an identifier was found.

0498: E Negative numeric literal is not permitted.

The context does not allow the use of a negative numeric literal.

0499: E Reference modification is not permitted here.

The context does not allow the use of a reference modification specification.

0500: E Leftmost-character-position or length in reference modifier exceeds length of data item.

The value of the indicated numeric literal is too large for its use in a reference modification specification. The offset and length values may not exceed the length of the data item being reference modified. If both the offset and length are specified as literals, their sum less 1 may not exceed the length of the data item being reference modified.

0501: E Source language feature not supported by specified version of runtime.

The indicated language feature is incompatible with the requested runtime support version level. The requested version level is determined by a compiler option.

0502: E Colon required after leftmost-character-position in reference modifier.

A colon separator is required following the left operand of a reference modification specification.

0504: E Unique data-name required when EXTERNAL or GLOBAL clause is present in data description entry.

The indicated clause may not be specified in the current data description entry since either the description does not specify a data-name (that is, is implicitly or explicitly FILLER) or the data-name is the same as another data-name described with the same clause.

0505: E Clause and level-number conflict.

The indicated clause clashes with the level-number of the data description entry in which it is specified. An EXTERNAL or GLOBAL clause may not be specified in a data description entry if the level-number is other than 01. An EXTERNAL clause may be specified in the FILE SECTION only in a file description (FD) entry. An OCCURS clause may be specified in a data description entry only when the level-number is 02 through 49.

0506: E Clause is not permitted when file-name specified in MULTIPLE FILE TAPE clause.

The indicated clause may not be specified in a file description entry for a file which is listed in any MULTIPLE FILE TAPE clause in the I-O-CONTROL paragraph.

0507: E Clause is not permitted when file-name specified in SAME AREA or SAME RECORD AREA clause.

The indicated clause may not be specified for the current file description entry or record description entry for a file since the file is listed in a SAME clause in the I-O-CONTROL paragraph. For the EXTERNAL clause, the file may not be listed in any SAME AREA, SAME RECORD AREA or SAME SORT AREA clause. For the GLOBAL clause, the file may not be listed in any SAME RECORD AREA clause.

0508: E EXTERNAL and GLOBAL clauses are only permitted in File and Working-Storage Sections.

The indicated clause may not be specified in the current section of the Data Division.

0509: E EXTERNAL and REDEFINES clauses not permitted in same data description entry.

The EXTERNAL clause is specified with the REDEFINES clause. These clauses are mutually exclusive within a single data description entry.

0511: E COMMON clause permitted only if program is contained within another program.

The indicated syntax is allowed only within a nested program, and the current program is not contained within another program.

0512: E LINE or COLUMN option required in AT phrase of ACCEPT or DISPLAY screen-name statement.

The AT keyword in a Format 5 ACCEPT statement or a Format 3 DISPLAY statement is not followed by LINE, COLUMN or COL.

0513: E Clause permitted only at elementary level in Screen Section.

The relation between the current level number and the preceding level number implies that the preceding item is a group, but the preceding item description includes attributes allowed only at the elementary level.

0514: E Color integer value must be in range 0 (black) through 7 (white).

The integer specified in a BACKGROUND-COLOR or FOREGROUND-COLOR clause is not in the range 0 to 7 as required.

0515: E Integer expected. Literal with digits to right of decimal point is not permitted here.

The context requires an integer numeric literal, but a numeric literal with digits to the right of the decimal point was found.

0519: E COLUMN NUMBER data item is wrong linkage item for screen-name: *screen-name-1*

The data-name specified in the COLUMN clause of the indicated Screen Section data item is defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, nor is it defined subordinate to such a data-name.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0520: E Repeated screen description clause is not permitted.

A Screen Section attribute has been specified more than once.

0526: E LINE NUMBER data item is wrong linkage item for screen-name: *screen-name-1*

The data-name specified in the LINE clause of the indicated Screen Section data item is defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, nor is it defined subordinate to such a data-name.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0527: E Screen-name expected.

If the first primary operand of a DISPLAY statement is a screen-name, all subsequent primary operands of that DISPLAY statement must also be screen-names.

0528: E Screen-name is not permitted here.

The screen-name is specified in the source program where a screen-name is not allowed. Screen-names may be specified only as certain operands of ACCEPT and DISPLAY statements.

0529: E Split-key-name is not permitted here.

The split-key-name is specified in the source program where a split-key-name is not allowed. Split-key-names may be specified only as certain operands of READ and START statements.

0533: E Subscript data item is wrong linkage item for screen-name: *screen-name-1*

The data-name specified as a subscript in the description of the indicated Screen Section data item is defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, nor is it defined subordinate to such a data-name.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0534: E FROM, TO, or USING phrase expected following PICTURE character-string in screen description entry.

In the Screen Section, the PICTURE character-string in a PICTURE clause must be followed by a TO, FROM or USING phrase.

0535: E PICTURE and VALUE clause not permitted in same screen description entry.

In the Screen Section, an item description cannot contain both a PICTURE and a VALUE clause.

0541: E USING/FROM data item is wrong linkage item for screen-name: *screen-name-1*

The data-name specified as a source item (FROM or USING) in the description of the indicated Screen Section data item is defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, nor is it defined subordinate to such a data-name.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0544: E SIZE phrase in START statement not permitted for relative file.

The SIZE phrase is specified in a START statement for a relative file. The SIZE phrase is meaningful only for indexed files.

0545: E SIZE phrase in START statement specifies integer value that exceeds key size.

The SIZE phrase specifies an integer value that is greater than the length of the key data item specified in the same START statement. The integer value must not exceed the key size since the SIZE phrase is meaningful only for limiting the comparison to a size less than or equal to the length of the key data item.

0546: W VALUE OF FILE-ID clause specifies different file access name than ASSIGN clause.

The file description entry VALUE OF FILE-ID clause specifies a different file access name literal or data-name than specified in the file control entry ASSIGN clause. The file access name should be specified in only one of these two alternatives, but if specified in both they must agree. The ASSIGN clause specification takes precedence when this warning occurs.

0547: E TO data item is wrong linkage item for screen-name: *screen-name-1*

The data-name specified as a target item (TO or USING) in the description of the indicated Screen Section data item is defined in the Linkage Section. The data-name is not listed in the Procedure Division USING phrase, nor is it defined subordinate to such a data-name.

Note If the object version is not restricted to less than 8, this message will not be produced under the condition described above. Instead, the item will be considered a based linkage item. Message 665 will occur if the base address of the linkage record is never set within the program.

0548: E Level-number 01 or 77 expected in Screen Section.

An entry in the Screen Section of the Data Division is neither a record description entry (level-number 01) nor a 77 level description entry (level-number 77).

0549: W Screen-name in ACCEPT statement has no input screen items.

A screen-name specified as an operand in an ACCEPT statement has no subordinate elementary fields that specify a TO or USING attribute.

0567: E Operand size must be one character.

The indicated operand must be a single character literal or refer to a single character data item.

0568: E Operand sizes must match.

The indicated operand and the preceding operand have different lengths but are required to be the same size.

0630: E COPY statement is not permitted within COPY statement.

The indicated COPY statement is embedded within another COPY statement. Such nesting of COPY statements is not permitted. This error may be the result of omitting the required period that should have been present to end the preceding COPY statement.

0631: E End of source file encountered while scanning REPLACING phrase of COPY statement.

The COPY statement is incomplete because the end of the current source file was encountered before the required period that ends the COPY statement was found. The COPY statement must be complete within one source file.

0633: W Screen description clause is not permitted with VALUE clause.

The screen description entry contains the VALUE clause and another clause that is mutually exclusive with the VALUE clause. The AUTO, BLANK WHEN ZERO, FULL, JUSTIFIED, PICTURE, REQUIRED, and SECURE clauses must not be specified in a screen description entry that also specifies the VALUE clause.

Note The VALUE clause may be implicitly specified by a nonnumeric literal.

0636: W Duplicate split keys are not permitted. Split-key-name is: *split-key-name-1*

The indicated split-key-name duplicates another key. That is, it has the same number of segments as another key and each segment has the same offset and length as the corresponding segments of the other split-key. Such duplicates are not allowed because they cannot be uniquely ordered and result in redundant indexes for the file.

0637: W No data items for this identifier are eligible for initialization.

The indicated identifier in an INITIALIZE statement has no data items that qualify for initialization according to the rules of the INITIALIZE statement.

0638: E Data-name required in RENAMES clause.

The user-defined word in the RENAMES clause must be a data-name, but another type of user-defined word was found.

0639: E Repeated file-name in MULTIPLE FILE TAPE clause is not permitted.

A given file-name may only be specified once in a MULTIPLE FILE TAPE clause but the indicated file-name has already been specified in a MULTIPLE FILE TAPE clause.

0640: W ALPHABETIC in INITIALIZE statement but there are no alphabetic data items in receiving items.

The INITIALIZE statement specifies the REPLACING ALPHABETIC DATA phrase but none of the receiving items are alphabetic data items or groups that contain alphabetic data items. Therefore, no initialization results from the specification of this phrase. When the REPLACING phrase is specified without the VALUE or DEFAULT phrases, only data items that belong to the category or categories specified in the REPLACING phrase are initialized.

0641: W ALPHANUMERIC in INITIALIZE statement but there are no alphanumeric data items in receiving items.

The INITIALIZE statement specifies the REPLACING ALPHANUMERIC DATA phrase but none of the receiving items are alphanumeric data items or groups that contain alphanumeric data items. Therefore, no initialization results from the specification of this phrase. When the REPLACING phrase is specified without the VALUE or DEFAULT phrases, only data items that belong to the category or categories specified in the REPLACING phrase are initialized.

0642: W NUMERIC in INITIALIZE statement but there are no numeric data items in receiving items.

The INITIALIZE statement specifies the REPLACING NUMERIC DATA phrase but none of the receiving items are numeric data items or groups that contain numeric data items. Therefore, no initialization results from the specification of this phrase. When the REPLACING phrase is specified without the VALUE or DEFAULT phrases, only data items that belong to the category or categories specified in the REPLACING phrase are initialized.

0643: W ALPHANUMERIC-EDITED in INITIALIZE statement but there are no alphanumeric edited data items in receiving items.

The INITIALIZE statement specifies the REPLACING ALPHANUMERIC-EDITED DATA phrase but none of the receiving items are alphanumeric edited data items or groups that contain alphanumeric edited data items. Therefore, no initialization results from the specification of this phrase. When the REPLACING phrase is specified without

the VALUE or DEFAULT phrases, only data items that belong to the category or categories specified in the REPLACING phrase are initialized.

0644: W NUMERIC-EDITED in INITIALIZE statement but there are no numeric edited data items in receiving items.

The INITIALIZE statement specifies the REPLACING NUMERIC-EDITED DATA phrase but none of the receiving items are numeric edited data items or groups that contain numeric edited data items. Therefore, no initialization results from the specification of this phrase. When the REPLACING phrase is specified without the VALUE or DEFAULT phrases, only data items that belong to the category or categories specified in the REPLACING phrase are initialized.

0645: W Program-name length exceeds 30 characters.

A program-name must not exceed 30 characters in length. Only the first 30 characters of the specified program-name are retained to identify the object program.

0651: W Negative literal moved to unsigned receiving data item; absolute value of literal used.

The indicated receiving data item in a MOVE statement is unsigned numeric, alphanumeric, or alphanumeric edited, but the sending item is a negative literal. The absolute value of the literal will be moved to the indicated receiving item. It is likely that a signed numeric receiving data item should be specified in the source program in place of the indicated receiving data item.

0652: W Negative literal in relation with unsigned data item; condition does not depend on data item value.

In the indicated relation condition, one operand is a negative numeric literal and the other operand is an unsigned numeric data item. Since a negative numeric literal is always less than an unsigned numeric value, the relation result is independent of the value of the data item. This probably indicates a coding error. Either the numeric data item should be described as signed or the literal should not be negative.

0653: W Sending nonnumeric literal value is not compatible with receiving numeric or numeric edited data item.

The indicated receiving item is a numeric or numeric edited data item. The sending item is a nonnumeric literal that contains characters other than decimal digits. COBOL defines such a move only when the sending item is a string of decimal digits representing a positive integer value. The literal must not contain a sign representation, decimal point, currency symbol, space, or comma. Note that the figurative constants, LOW-VALUES and HIGH-VALUES, are normally not valid sending items for a numeric or numeric edited receiving item. Only when the program collating sequence is defined such that LOW-VALUES or HIGH-VALUES represent a decimal digit is such a move defined in COBOL.

0657: E Data item has zero size or group is not yet completed. Value of 0 assumed.

For the indicated constant-expression LENGTH OF *data-name-1* operator, the referenced *data-name-1* has zero length at the time the operator is evaluated. The common cause of this error is specifying a *data-name-1* for a group that has not been allocated yet because a level-number that is less than or equal to the level-number of *data-name-1* has not yet been scanned. The level-number 78 data description entry containing this reference must be moved after a data description entry with a level-number less than or equal to the level-number used in the description of *data-name-1*.

0662: E Symbol table size exceeds capacity of object version 7; object version has been forced to 8.

The Compile Command Option Z or the OBJECT-VERSION keyword of the COMPILER-OPTIONS configuration record has been used to force the object version to less than 8, the Compile Command Option Y or the SYMBOL-TABLE-OUTPUT keyword of the COMPILER-OPTIONS configuration record has been specified to indicate that the symbol table should be written to the object file for debugging purposes, and the program used more than 64K of name space. Object versions less than 8 did not support a debugging symbol table that required more than 64K of name space. The compiler forces the object version to 8 so that the symbol table can be properly output to the object file.

0663: E Pointer data item is not permitted here.

A pointer data item, literal, or special register has been used where it is not permitted. Pointer items may only be used in the VALUE clause of a data description entry that specifies USAGE IS POINTER, in a relation condition with another pointer item, in the USING and GIVING phrases of the Procedure Division header or of the CALL statement, or in Formats 5 and 6 of the SET statement.

0664: E Pointer data item expected here.

The indicated context requires a pointer item, but the item found by the compiler is not a pointer data item, an ADDRESS special register, nor the figurative constant NULL (NULLS).

0665: E The base address was never set for the referenced based linkage record: *data-name-1*

The indicated *data-name-1* is defined in the Linkage Section and is not a formal argument of the program, that is, it is not listed in the USING or GIVING phrases of the Procedure Division header, nor is it a redefinition or renaming of such an item. *data-name-1* or a data item subordinate to it has been referenced in the program, but the base address of the record has never been set with a Format 5 SET statement. If the statement that made the reference were executed during the run unit, the run unit would be terminated with data reference error 108 since the record will have a null base address.

0666: E Special register not permitted here. An identifier is required.

A special register has been referenced where an identifier is required, and the special register is not one that can be a receiving operand.

0669: E Reserved word *"reserved-word"* expected.

The context requires a specific reserved word or one of a specific set of words. The required word or one of the set of words is given in quotation marks in the message text.

0670: E Statement is permitted only within a paragraph.

The indicated statement requires a paragraph, but no paragraph-name was specified at the beginning of the Procedure Division.

0671: E Statement is permitted only within an in-line PERFORM statement.

The indicated statement requires an in-line PERFORM statement, but is specified outside of any in-line PERFORM statement.

0672: E Statement is permitted only within a section.

The indicated statement requires a section, but is not specified within a section.

0673: E LIKE conditional variable must be a nonnumeric data item.

The conditional variable data item for which a level-number 88 condition-name is associated that uses the LIKE relational operator is described as a numeric data item. The conditional variable in this case must be described as a nonnumeric data item.

0674: E LIKE condition subject must be a nonnumeric data item or literal.

The subject of a LIKE relation condition must be a nonnumeric data item or nonnumeric literal. The subject of a LIKE condition must not be a numeric data item or numeric literal.

0675: E LIKE condition subject must not be a pointer data item.

The subject of a LIKE relation condition must not be a pointer data item.

0676: E LIKE condition pattern must be nonnumeric data item, nonnumeric literal, or pointer data item.

The pattern of a LIKE relation condition must be a nonnumeric data item, nonnumeric literal or a pointer data item. The pattern of a LIKE relation condition must not be a numeric data item or a numeric literal.

0677: E Numeric to nonnumeric relation requires the numeric subject to be DISPLAY usage.

The indicated relation condition compares a numeric value to a nonnumeric value, but the numeric subject is not DISPLAY usage as required for such a relation condition.

0678: E Numeric to nonnumeric relation requires the numeric subject to be an integer.

The indicated relation condition compares a numeric value to a nonnumeric value, but the numeric subject is not an integer as required for such a relation condition.

0679: E Relation object must not be a pointer when relation subject is not a pointer.

The indicated relation condition compares a non-pointer value to a pointer value, but only a pointer value may be compared to another pointer value.

0680: E Relation subject must not be a pointer when relation object is not a pointer.

The indicated relation condition compares a pointer value to a non-pointer value, but a pointer value may only be compared to another pointer value.

0681: E Table element requires subscripting before reference modification.

Reference modification has been specified for a table element without the required subscripting specification. The subscripting for a table element data item must be supplied first (leftmost) and then reference modification, if desired, may be specified in a COBOL identifier.

0682: E Pattern class character range cannot include multi-character escape.

A class character range of the form ***s-e*** in a pattern cannot specify a multi-character escape for either ***s*** or ***e***. A multi-character escape specifies a set of matching characters and thus is not allowed as the starting or ending point of a class character range of the form ***s-e***.

0683: E Pattern class character range cannot be hyphen '-' except at beginning or end of positive character group.

Within a character class expression of a pattern, a hyphen cannot be used to represent itself except at the beginning or end of a positive character range. This is necessary to allow the hyphen to be interpreted as specifying a character range of the form ***s-e*** or character class subtraction.

0684: E Pattern class character range cannot be opening bracket '['.

Within a character class expression of a pattern, the opening bracket character '[' may not be used as a character range character. To include an opening bracket, it must be escaped as '\['.

0685: E Pattern class character range cannot specify decreasing range.

Within a character class expression of a pattern, a character range of the form ***s-e*** must specify an increasing range. That is, the character range character ***e*** must not be less than the character range character ***s***.

0686: E Pattern character class subtraction cannot be followed by additional class specification.

Within a character class expression of a pattern, a character class subtraction must be specified last (rightmost) in the expression. The closing bracket of the character class expression containing the subtraction expression must immediately follow the subtraction expression.

0687: E Pattern escape sequence (initiated by '\') is not valid.

Within a pattern regular expression, an escape sequence initiated by a backslash ('\') is not a valid single-character escape, multi-character escape, or category escape. The indicated escape sequence is reserved for possible future definition and, therefore, is not currently defined nor allowed.

0688: E Pattern compilation requires more memory than is available.

The pattern regular expression is either too large or too complex to compile in the amount of memory available. The pattern must be made smaller or simpler. It is possible that making more memory available to the pattern compilation process by embedding the pattern in a smaller or simpler COBOL program will allow the original pattern to compile.

0689: E Pattern quantifier opened with '{' is missing the closing brace '}'.

The pattern regular expression contains a quantifier opened with a brace ('{'), but the required closing brace ('}') is not present.

0690: E Pattern character class expression is missing the closing bracket ']'.

The pattern regular expression contains a character class expression opened with a bracket (‘[’), but the required closing bracket (‘]’) is not present.

0691: E Pattern parenthesized subexpression is missing the closing parenthesis ')'.

The pattern regular expression contains a parenthesized subexpression opened with a parenthesis (‘(’), but the required closing parenthesis (‘)’) is not present.

0692: E Pattern category escape '\p{' or '\P{' is missing the closing brace '}'.

The pattern regular expression contains a category escape, but the required closing brace (‘}’) is not present.

0693: E Pattern category escape '\p{' or '\P{' is missing the opening brace '{'.

The pattern regular expression contains a category escape, but the required opening brace (‘{’) is not present.

0694: E Pattern category escape '\p{' or '\P{' contains an unknown category specification.

The pattern regular expression contains a category escape, but the category specification provided between the braces is not recognized. Valid category escape specifications consist of one uppercase letter followed, optionally, by one lowercase letter. Only certain combinations are defined and allowed, as specified in the documentation for a pattern regular expression. Additionally, category escapes may specify a block escape of the form `Is`*`BlockName`*, where *`BlockName`* is the Unicode block name (with all white space stripped out) of a block of characters.

0695: E Pattern quantifier maximum count is less than the minimum count.

Within a pattern regular expression, a quantifier of the form “{*n*,*m*}” is specified where *m*, the maximum count, is less than *n*, the minimum count. The maximum count must not be less than the minimum count.

0696: E Pattern quantifier maximum count is missing; at least one decimal digit was expected.

Within a pattern regular expression, a quantifier of the form "{*n,m*}" is specified where *m*, the maximum count, has no decimal digits. Possibly, a quantifier of the form "{*n,*}" was intended and the closing brace was incorrectly entered.

0697: E Pattern quantifier maximum count is too large (> 65535).

Within a pattern regular expression, a quantifier of the form "{*n,m*}" is specified where *m*, the maximum count, is greater than 65535. The maximum count must be less than or equal to 65535.

0698: E Pattern quantifier minimum count is missing; at least one decimal digit was expected.

Within a pattern regular expression, a quantifier of the form "{*n*}", "{*n,*}" or "{*n,m*}" is specified where *n*, the minimum or exact count, has no decimal digits. Possibly, a quantifier of the form "{*n*}" or "{*n,*}" was intended and the closing brace or comma was incorrectly entered.

0699: E Pattern quantifier minimum count is too large (> 65535).

Within a pattern regular expression, a quantifier of the form "{*n*}", "{*n,*}" or "{*n,m*}" is specified where *n*, the minimum or exact count, is greater than 65535. The minimum count must be less than or equal to 65535.

0700: E Pattern contains an unexpected closing brace '}'.

Within a pattern regular expression, an unexpected closing brace was encountered. If a closing brace is intended as a match character, it must be specified as the single-character escape '\}'. Otherwise, a corresponding opening brace for a quantifier is required prior to a closing brace.

0701: E Pattern contains an unexpected closing bracket ']'.

Within a pattern regular expression, an unexpected closing bracket was encountered. If a closing bracket is intended as a match character, it must be specified as the single-character escape '\]'. Otherwise, a corresponding opening bracket for a character class expression is required prior to a closing bracket.

0702: E Pattern contains an unexpected closing parenthesis ')'.

Within a pattern regular expression, an unexpected closing parenthesis was encountered. If a closing parenthesis is intended as a match character, it must be specified as the single-character escape '\)'. Otherwise, a corresponding opening parenthesis for a parenthesized subexpression is required prior to a closing parenthesis.

0703: E Pattern contains an unexpected quantifier '*' that is not preceded by a valid atom.

Within a pattern regular expression, an unexpected asterisk was encountered. If an asterisk is intended as a match character, it must be specified as the single-character escape '*'. Otherwise, a valid atom of the regular expression must precede the asterisk. If the atom already specifies another quantifier, the atom must be parenthesized before another quantifier may be applied.

0704: E Pattern contains an unexpected quantifier '+' that is not preceded by a valid atom.

Within a pattern regular expression, an unexpected plus sign was encountered. If a plus sign is intended as a match character, it must be specified as the single-character escape '\+'. Otherwise, a valid atom of the regular expression must precede the plus sign. If the atom already specifies another quantifier, the atom must be parenthesized before another quantifier may be applied.

0705: E Pattern contains an unexpected quantifier '?' that is not preceded by a valid atom.

Within a pattern regular expression, an unexpected question mark was encountered. If a question mark is intended as a match character, it must be specified as the single-character escape '\?'. Otherwise, a valid atom of the regular expression must precede the question mark. If the atom already specifies another quantifier, the atom must be parenthesized before another quantifier may be applied.

0706: E Pattern contains an unexpected quantifier '{' that is not preceded by a valid atom.

Within a pattern regular expression, an unexpected opening brace was encountered. If an opening brace is intended as a match character, it must be specified as the single-character escape '\{'. Otherwise, a valid atom of the regular expression must precede the opening brace. If the atom already specifies another quantifier, the atom must be parenthesized before another quantifier may be applied.

0707: E Pattern is too large or complex to compile.

The pattern regular expression is either too large or too complex to compile because it generates a state machine description that is greater than 65535 bytes in length, which is the maximum supported by the implementation. The pattern must be made smaller or simpler. Patterns on the order of 5,000 to 10,000 characters in length should compile without exceeding this limit.

0720: E Relational operator specified with first condition value does not define a true value.

A true value cannot be determined for use in the SET statement because the specified level-number 88 condition-name is defined with a relational operator for the first value given in the associated Format 2 VALUE clause and that relational operator does not define a true value. Only relational operators that include an equality relation define a true value. Thus, the relational operators NOT EQUAL, LESS THAN, GREATER THAN, and LIKE, when used with the first value in the Format 2 VALUE clause, do not define a true value for the condition-name. The true value may be defined by listing it first in the Format 2 VALUE clause without a relational operator or by using a relational operator that includes an equality relation such as EQUAL, NOT LESS THAN, or NOT GREATER THAN.

0721: W DATA-POINTER in INITIALIZE statement but there are no data pointer data items in receiving items.

The INITIALIZE statement specifies the REPLACING DATA-POINTER DATA phrase but none of the receiving items are data pointer data items or groups that contain data pointer data items. Therefore, no initialization results from the specification of this phrase. When the REPLACING phrase is specified without the VALUE or DEFAULT phrases, only data items that belong to the category or categories specified in the REPLACING phrase are initialized.

0722: E One or more data categories as described for category-name are required here.

The REPLACING phrase has been specified in an INITIALIZE statement, but is not followed by a recognized category from the category-name list. The REPLACING phrase of the INITIALIZE statement requires that one or more categories be specified.

0723: W Repeated category in INITIALIZE statement category-name list is not permitted.

A category-name construct in the VALUE or REPLACING phrases of the INITIALIZE statement permits only unique categories. The compiler ignores, after producing this message, a repeated category within one category-name construct.

0724: W FILLER phrase conflicts with configured suppression of FILLER identifiers in symbol table.

The FILLER phrase in the INITIALIZE statement has been specified, but the source program is being compiled with SUPPRESS-FILLER-IN-SYMBOL-TABLE=YES configured in the COMPILER-OPTIONS configuration record. When FILLER data items are suppressed from the symbol table, they are not available to the compiler for reference by the FILLER phrase of the INITIALIZE statement.

Appendix C: Glossary

The terms in this appendix are defined in accordance with their meaning in RM/COBOL, and may not have the same meaning for other languages.

These definitions are also intended as either reference or introductory material to be reviewed prior to reading the detailed language specifications. For this reason, these definitions are, in most instances, brief and do not include detailed syntactical rules. Complete specifications for elements defined in this section can be located in the chapters of this document.

Definitions

66-Level-Description-Entry. A data description entry with the level-number 66 that describes a data item as a renaming of previously described data items.

77-Level-Description-Entry. A data description entry with the level-number 77 that describes a noncontiguous data item with the level-number 77.

78-Level-Description-Entry. A data description entry with the level-number 78 that describes a constant-name.

88-Level-Description-Entry. A data description entry with the level-number 88 that describes a condition-name.

Abbreviated Combined Relation Condition. The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

Access Mode. The manner in which records are to be operated upon within a file. COBOL supports three access modes: sequential, random, and dynamic.

Actual Argument. A data item named in the USING or GIVING phrases of a CALL statement. Both the calling and the called program may refer to these data items. The called program refers to the actual argument by using the name of the corresponding formal argument.

Actual Decimal Point. The physical representation, using the decimal point character period (.) or comma (,), of the decimal point position in a data item.

Alphabetic Character. A letter or a space character.

Alphabet-Name. A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, that assigns a name to a specific character set, collating sequence, or both.

Alphanumeric Character. Any character in the character set of the computer.

Alternate Record Key. A key, other than the prime record key, whose contents identify a record within an indexed file.

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

Arithmetic Operation. The process caused by the execution of an arithmetic statement, or the evaluation of an arithmetic expression, that results in a mathematically correct solution to the arguments presented.

Arithmetic Operator. A single character or fixed two-character combination which belongs to the following set:

Character	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Arithmetic Statement. A statement that causes an arithmetic operation to be run. The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements.

Ascending Key. A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with the rules for comparing data items.

Assumed Decimal Point. A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning with no physical representation.

At End Condition. A condition caused:

1. During the running of a READ statement for a sequentially accessed file, when no next logical record exists in the file, or when the number of significant digits in the relative record number is larger than the size of the relative key data item, or when an optional input file is not present.

2. During the running of a RETURN statement, when no next logical record exists for the associated sort or merge file.

3. During the running of a SEARCH statement, when the search operation ends without satisfying the condition specified in any of the associated WHEN phrases.

Automatic Multiple. A record locking mode in which the READ statement executed in shared input-output mode automatically obtains a lock on the record accessed except when the NO LOCK phrase is specified. Multiple record locks for the logical file may be held by the run unit. The record locks are released only by execution of a CLOSE or UNLOCK statement, except that the successful execution of a DELETE statement releases the lock on the deleted record.

Automatic Record Locking Modes. Record locking modes in which the READ statement executed in shared input-output mode automatically obtains a lock on the record accessed except when the NO LOCK phrase is specified. The automatic record locking modes are automatic single and automatic multiple.

Automatic Single. A record locking mode in which the READ statement executed in shared input-output mode automatically obtains a lock on the record accessed except when the NO LOCK phrase is specified. Only a single record for the logical file is locked by the run unit since the next input-output operation on the file releases any existing record lock.

Based Linkage Record. A record-description-entry or 77-level-description-entry described in the Linkage Section that receives its base address by use of a Format 5 SET statement. Linkage records are not allocated storage during compilation. A based linkage record is assigned (Format 5) or reassigned (Format 6) to a storage location by use of the SET statement that specifies the ADDRESS OF *data-name-1* as the receiving item, where *data-name-1* names the based linkage record. Based linkage records may include formal arguments of the program.

Binary Allocation Override. An integer specified in parentheses following one of the binary usage words BINARY, COMP-4, or COMPUTATIONAL-4 in the USAGE clause. The integer must be in the range one through sixteen and specifies the number of character positions (bytes) of storage to allocate for the binary data item being described. A binary allocation override may also follow COMP or COMPUTATIONAL if the compiler has been configured for treating this usage type as binary.

Binary Sequential. A record delimiting technique that allows packed-decimal and binary data items in the record. For fixed-length record files, no record delimiter is needed or used. For variable-length record files, a record length header and trailer is stored with the record on the external storage medium.

Block. A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with physical record.

Bottom Margin. An empty area which follows the page body.

Called Program. A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit.

Calling Program. A program which starts a CALL to another program.

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.

Channel-Name. A feature-name that names a channel on a printer carriage control tape or 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 whose usage is DISPLAY. (See Appendix C, *Internal Data Formats*, of the *RM/COBOL User's Guide* for further characteristics of physical storage.)

Character-String. A sequence of adjacent characters which form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

Class Condition. The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric or consists exclusively of those characters listed in the definition of a class-name.

Class-Name. A user-defined word defined in the SPECIAL-NAMES paragraph of the Environment division that assigns a name to the proposition for which a truth value can be defined, that the content of a data item consists exclusively of those characters listed in the definition of the class-name.

Clause. A clause is an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL Character Set. The complete COBOL character set consists of the characters listed below.

Character	Meaning
0, 1, . . . , 9	Digit
A, B, . . . , Z	Uppercase letter
a, b, . . . , z	Lowercase letter
	Space
+	Plus sign
–	Minus sign (hyphen)
*	Asterisk
/	Slant (solidus)
=	Equal sign
$	Currency sign (represented as ¤ in the International Reference Version of International Standard, ISO 646-1973)
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point, full stop)
“	Quotation mark (double quotation)
’	Apostrophe (single quotation)
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol
:	Colon

Note When the computer character set includes lowercase letters, they may be used in character-strings. Except when used in nonnumeric literals and some PICTURE symbols, each lowercase letter is equivalent to the corresponding uppercase letter.

COBOL Word. A character-string of not more than 30 characters which forms a user-defined word, a system-name, a context-sensitive word, or a reserved word.

Code-Name. A system-name that names a character code set or collating sequence or both.

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 or screen line. The columns are numbered from 1, by 1, starting at the farthest left character position of the line and extending to the farthest right position of the line.

Combined Condition. A condition that is the result of connecting two or more conditions with the 'AND' or the 'OR' logical operator.

Comment Line. A source program line represented by an asterisk (*) in the indicator area of the line and any characters from the character set of the computer 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 character set of the computer in area A and area B of that line causes page ejection prior to printing the comment.

Comment-Entry. An entry in the Identification Division that may be any combination of characters from the character set of the computer.

Common Program. A program which, despite being directly contained within another program, may be called from any program directly or indirectly contained in that other program.

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. It describes the interface between the message control system (MCS) and the COBOL program.

Communication Device. A mechanism, hardware or hardware plus 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 message control system (MCS) and the program, composed of one or more communication description areas.

Compile Time. The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL 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 compiler directing statements are the COPY, ENTER, REPLACE, and USE statements.

Complex Condition. A condition in which one or more logical operators act upon one or more conditions.

Composite of Operands. A hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points. This data item must not contain more than 30 decimal digits.

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 execution time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2, . . .) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2, . . .) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

Conditional Expression. A simple condition or a complex condition specified in an EVALUATE, IF, PERFORM, or SEARCH statement.

Conditional Phrase. A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from the execution of a conditional statement.

Conditional Statement. A 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. (Contrast with "Imperative Statement.")

Conditional Variable. A data item one or more values of which has a condition-name assigned to it.

Condition-Name. A user-defined word that assigns a name to a subset of values that a conditional variable may assume; or a user-defined word assigned to a status of an implementor-defined switch or device. When 'condition-name' is used in the general formats, it represents a unique data item reference consisting of a syntactically correct combination of a condition-name, together with qualifiers and subscripts, as required for uniqueness of reference.

Condition-Name Condition. The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

Configuration Section. A section of the Environment Division that describes overall specifications of source and object programs.

Constant-Expression. A constant-name that has already been defined with an integer value other than the one being defined in the current 78-level-description-entry, a numeric integer literal, a NEXT, LENGTH OF, or START OF, or DATE-COMPILED operator, such constant-names, literals and operators preceded by the constant-expression operator NOT, such constant-names, literals and operators separated by constant-expression operators (+, -, *, /, **, AND, OR, EXCLUSIVE OR), two constant expressions separated by a constant-expression operator, or a constant-expression enclosed in parentheses. Constant-expressions are evaluated in strict left to right order with no precedence other than expressions within parentheses are evaluated first.

Constant-Name. A user-defined word that assigns a name to a literal value in a level-number 78 data description entry. After the constant-name is defined, it may be used wherever a literal is shown in the general formats unless otherwise prohibited. A constant-name with an integer literal value may also be used wherever an integer, level-number, or segment-number is shown in the general formats. A constant-name with an integer value may be used as the repeat count in a PICTURE character-string. The effect of a constant-name reference is the same as if the literal value assigned to the constant-name were written instead.

Context-Sensitive Word. A COBOL word that is reserved in a specified language construct or context. If a context-sensitive word is used where the context-sensitive word is permitted in the specified language construct or context, the word is treated as a keyword; otherwise, it is treated as a user-defined word.

Contiguous Items. Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchical 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.

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 which is available in the record area associated with a file.

Current Volume Pointer. A conceptual entity that points to the current volume of a sequential file.

Data Clause. A clause, specified 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, condition-name, or constant-name, if required, and then followed by a set of data clauses, as required.

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 description entry. When used in the general formats, 'data-name' represents a word which must not be reference-modified, subscripted, or qualified unless specifically permitted by the rules of the format.

Debugging Line. A debugging line is any line with a 'D' in the indicator area of the line.

Declarative Sentence. A compiler directing sentence consisting of a single USE statement stopped by the separator period.

Declaratives. A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one, or more associated paragraphs.

De-Edit. The logical removal of all editing characters from a numeric edited data item in order to determine the unedited numeric value of the item.

Delimited Scope Statement. Any statement which includes its explicit scope terminator.

Delimiter. A character or a sequence of adjacent characters that identify the end of a string of characters and separates 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.

Device Name. A system-name that names a class of input-output devices. Each class is characterized by the statements, open modes, and file organizations it supports.

Digit Position. A digit position is 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. If the data description entry specifies that usage is DISPLAY, then a digit position is synonymous with a character position. (See Appendix C, *Internal Data Formats*, of the *RM/COBOL User's Guide* for further characteristics of physical storage.)

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 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 as follows:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [USING {data-name-1} ... ].
```

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. (Compare with definitions for "Sequential Access" and "Random Access.")

Editing Character. A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
B	Space
0	Zero
+	Plus
–	Minus
CR	Credit
DB	Debit
Z	Zero suppress
*	Check protect
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)
/	Slant (solidus)

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. The end program header is:

```
END PROGRAM [program-name].
```

Entry. Any descriptive set of consecutive clauses ended 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.

Exclusive File. A file that is open with a lock mode of exclusive. An exclusive input-output or output file may not be open concurrently by any other run unit. An exclusive input file may not be open concurrently by any other run unit except in the input mode.

Exclusive Mode. A lock mode in which, for extend, input-output and output modes, access to the file is denied to any other run unit and, for input mode, access is denied to any other run unit that attempts to open the file for extend or input-output mode. A file cannot be successfully opened in exclusive mode if any other run unit has the file open in a conflicting mode.

Execution Time. The time at which an object program is run. The term is synonymous with object time.

Explicit Scope Terminator. A reserved word that ends the scope of a particular Procedure Division statement.

Expression. An arithmetic or conditional expression.

Extend Mode. The state of a file after running an OPEN statement, with the EXTEND phrase specified, for that file and before running a CLOSE statement without the REEL or UNIT phrase for that file.

External Data. The data described in a program as external data items and external file connectors.

External Data Item. A data item that is described as part of an external record in one or more programs of a run unit and which itself may be referred to from any program in which it is described.

External Data Record. A logical record that is described in one or more programs of a run unit and whose constituent data items may be referred to from any program in which they are described.

External File Connector. A file connector which is accessible to one or more object programs in the run unit.

External Switch. A hardware or software device, defined and named by the implementor, which is used to indicate that one of two alternate states exists.

Feature-Name. A system-name that names a channel on a printer carriage control tape or program.

Figurative Constant. A compiler generated value referred through the use of certain reserved words.

File. A collection of logical records.

File Access Name. The name communicated to the operating system to identify a physical file. The file access name may be explicitly specified in the ASSIGN clause of the file control entry or the VALUE OF FILE-ID clause of the file description entry. If not explicitly specified, the file access name defaults to the COBOL file-name. The runtime system may further modify the file access name before it is communicated to the operating system as explained in the *RM/COBOL User's Guide*.

File Attribute Conflict Condition. An unsuccessful attempt has been made to run an input-output operation on a file and the file attributes, as specified for that file in the program, do not match the fixed attributes for that file.

File Clause. A clause that appears as part of any of the following Data Division entries: file description entry (FD entry) and sort-merge file description entry (SD entry.)

File Connector. A storage area which contains information about a file and is used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

File Control Entry. A SELECT clause and all its subordinate clauses which declare the relevant physical attributes of a file.

File Description Entry. An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

File Organization. The permanent logical file structure established at the time that a file is created.

File Position Indicator. A conceptual entity that contains the value of the current key within the key of reference for an indexed file, or the record number of the current record for a sequential file, or the relative record number of the current record for a relative file, or indicates that no next logical record exists, or that the number of significant digits in the relative record number is larger than the size of the relative key data item, or that an optional input file is not present, or that the at end condition already exists, or that no valid next record has been established.

File Section. The section of the Data Division that contains file description entries and sort-merge file description entries together with their associated record descriptions.

FILE-CONTROL. The name of an Environment Division paragraph in which the data files for a given source program are declared.

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.

Fixed File Attributes. Information about a file which is established when a file is created and cannot subsequently be changed during the existence of the file. These attributes include the organization of the file (sequential, relative, or indexed), the prime record key, the alternate record keys, the minimum and maximum record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the minimum and maximum physical record size, the padding character, and the record delimiter.

Fixed-Length Record. A record associated with a file whose file description or sort-merge description entry requires that all records contain the same number of character positions.

Footing Area. The position of the page body adjacent to the bottom margin.

Formal Argument. A record-description-entry or 77-level-description-entry described in the Linkage Section that is named in the USING or GIVING phrases of the Procedure Division header. Formal arguments describe data items available from a calling program. Formal arguments are linkage records that receive their base address from the actual arguments passed in a CALL statement. The called program may override the actual argument address by using Format 5 of the SET statement, effectively converting the formal argument to a based linkage record.

Format. A specific arrangement of a set of data.

Global Name. A name that is declared in only one program but which may be referred to from that program and from any program contained within that program. Condition-names, data-names, file-names, record-names, and some special registers may be global names.

Group Item. A data item that is composed of subordinate data items.

High Order End. The farthest left character of a string of characters.

Identifier. A syntactically correct combination of a data-name, with its qualifiers, subscripts, and reference modifiers, as required for uniqueness of reference, that names a data item. The rules for ‘identifier’ associated with the general formats may, however, specifically prohibit qualification, subscripting, or reference modification.

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). The imperative verbs are listed in Chapter 1, *Language Structure*. An imperative statement may consist of a sequence of imperative statements. (Contrast with "Conditional Statement.")

Implicit Scope Terminator. A separator period which ends the scope of any preceding unterminated statement, or a phrase of a statement which by its occurrence indicates the end of the scope of any statement contained within the preceding phrase.

Index. A computer storage area or register, the content of which represents the identification of a particular element in a table.

Index Data Item. A data item in which the values associated with an index-name can be stored in a machine dependent form.

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. (Compare with definitions for "Relative Organization" and "Sequential Organization.")

Index-Name. A user-defined word that names an index associated with a specific table.

Initial Program. A program that is placed into an initial state every time the program is called in a run unit.

Initial State. The state of a program when it is first called in a run unit.

In-Line Comment. An in-line comment begins with the two contiguous characters *> preceded by a separator space, and ends with the last character position of the line. The in-line comment serves only for documentation in a program. The characters following the *> may be any characters from the character-set of the computer.

Input File. A file that is opened in the input mode.

Input Mode. The state of a file after running an OPEN statement, with the INPUT phrase specified, for that file and before running a CLOSE statement without the REEL or UNIT phrase for that file.

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.

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 which provides information required for transmission and handling of data during running of the object program.

Input-Output Statement. A statement that causes files to be processed by performing operations upon individual records or upon the file as a unit. The input-output statements are: ACCEPT, CLOSE, DELETE, DELETE FILE, DISABLE, DISPLAY, ENABLE, OPEN, PURGE, READ, RECEIVE, REWRITE, SEND, SET (with the TO ON or TO OFF phrase), START, UNLOCK, and WRITE.

Integer. A numeric literal or a numeric data item that does not include any digit position 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 Data Item. A data item which is described in one program in a run unit. An internal data item may have a global name.

Internal File Connector. A file connector which is accessible to only one object program in the run unit.

Intra-Record Data Structure. The entire collection of groups and elementary data items from a logical record which is defined by an adjacent subset of the data description entries which describe that record. These data description entries include all entries whose level-number is greater than the level-number of the first data description entry describing the intra-record data structure.

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 Mode. The state of a file after running an OPEN statement, with the I-O phrase specified, for that file and before running a CLOSE statement without the REEL or UNIT phrase for that file.

I-O Status. A conceptual entity which contains the two-character value indicating the resulting status of an input-output operation. This value is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

I-O-CONTROL. The name of an Environment Division paragraph in which object program requirements for rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

I-O-CONTROL Entry. An entry in the I-O-CONTROL paragraph of the Environment Division which contains clauses which provide information required for the transmission and handling of data on named files during the running of a program.

Key. A data item which 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.

Keyword. 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.

Letter. A character belonging to one of the following two sets:

1. uppercase letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z;

2. lowercase letters: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.

Level Indicator. Two letters 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 one or two digit number, which indicates the hierarchical position of a data item or the special properties of a data description entry. Level-numbers in the range 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, 78, and 88 identify special properties of a data description entry.

Library Text. A sequence of text words, comment lines, the separator space, or the separator pseudo-text delimiter in a COBOL library.

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.

LIKE Relation Condition. A special case of a relation condition that matches a subject data item to a pattern specified by a regular expression.

LINAGE-COUNTER. A special register whose value points to the current position within the page body.

Line Sequential. A record delimiting technique that matches the technique used by the standard system editor. In most systems this technique uses a sequence of one or more control characters appended to the record on the external storage medium. Therefore, files using this technique and containing packed decimal or binary data items cannot be reliably decomposed into the original output records during input.

Linkage Section. The section in the Data Division that describes formal arguments and based linkage records. Data description entries in the Linkage Section are not allocated storage during compilation, but describe data items available from a calling program or from having their base address set or modified by Formats 5 or 6 of the SET statement. All data items described in the Linkage Section initially have a null base address. In most cases, a reference to a Linkage Section data item will terminate the run unit with a data reference error unless the base address has been changed to a valid address by one of the following means:

- The data item is a formal argument, or is subordinate to a formal argument, that received a base address from an actual argument in a calling program.
- The data item is a based linkage record, or is subordinate to a based linkage record, for which the base address has been set to a value other than NULL by Format 5 of the SET statement.

Literal. A character-string whose value is implied by the ordered set of characters comprising the string.

Lock Mode. The manner in which a file is to be protected from concurrent access by other run units. RM/COBOL supports lock modes of exclusive and shared. For a shared input-output file, automatic multiple, automatic single, manual multiple or manual single record locking will apply.

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 Page. A conceptual entity consisting of the top margin, the page body, and the bottom margin.

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 item. When not further qualified, the term record refers to a logical record.

Low Order End. The farthest right character of a string of characters.

Low-Volume-I-O-Name. A system-name that names a low volume input-output device.

Manual Multiple. A record locking mode in which only a READ statement that specifies the LOCK phrase and is executed in shared input-output mode obtains a lock on the record accessed. Multiple record locks for the logical file may be held by the run unit. The record locks are released only by execution of a CLOSE or UNLOCK statement, except that the successful execution of a DELETE statement releases the lock on the deleted record.

Manual Record Locking Modes. Record locking modes in which only a READ statement that specifies the LOCK phrase and is executed in shared input-output mode obtains a lock on the record accessed. The manual record locking modes are manual single and manual multiple.

Manual Single. A record locking mode in which only a READ statement that specifies the LOCK phrase and is executed in shared input-output mode obtains a lock on the record accessed. Only a single record for the logical file is locked by the run unit since the next input-output operation on the file releases any existing record lock.

Mass Storage. A storage medium in which data may be organized and maintained in both a sequential and nonsequential manner.

Mass Storage Control System (MSCS). An input-output control system that directs, or controls, the processing of mass storage files.

Mass Storage File. A collection of records that is assigned to a mass storage medium.

MCS (Message Control System). A communication control system that supports the processing of messages.

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.

Message Control System (MCS). A communication control system that supports the processing of messages.

Message Count. The count of the number of complete messages that exist in the designated queue of messages.

Message Indicators. EGI (end of group indicator), EMI (end of message indicator), and ESI (end of segment indicator) are conceptual indications that serve to notify the message control system that a specific condition exists (end of group, end of message, or 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. Therefore, a message segment may be terminated by an ESI, EMI, or EGI; or, 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.

Mnemonic-Name. A user-defined word that is associated in the Environment Division with a specific feature-name, switch-name, or low-volume-I-O-name.

MSCS (Mass Storage Control System). An input-output control system that directs, or controls, the processing of mass storage files.

Multiple Record Locking Modes. Record locking modes in which locked records are not unlocked except by the explicit execution of a CLOSE or an UNLOCK statement that refers to the file. Multiple records may be locked in the file by the run unit. The multiple record locking modes are automatic multiple and manual multiple.

Native Character Set. The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

Native Collating Sequence. The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

Negated Combined Condition. The 'NOT' logical operator immediately followed by a parenthesized combined condition.

Negated Simple Condition. The 'NOT' logical operator immediately followed by a simple condition.

Next Executable Sentence. The next sentence to which control will be transferred after running of the current statement is complete.

Next Executable Statement. The next statement to which control will be transferred after running of the current statement is complete.

Next Record. The record which logically follows the current record of a file.

Noncontiguous Item. Elementary data items, in the Working-Storage and Linkage Sections, which 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 character set of the computer. 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 character set of the computer.

Null. The state of a pointer indicating that it contains no address.

Numeric Character. A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric Item. A data item whose description restricts its content to a value represented by characters chosen from the digits '0' through '9'; if signed, the item may also contain a '+', '–', or other representation of an operational sign.

Numeric Literal. A literal composed of one or more numeric characters that may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the farthest right character. The algebraic sign, if present, must be the farthest left character.

Object Computer Entry. An entry in the OBJECT-COMPUTER paragraph of the Environment Division which contains clauses which describe the computer environment in which the object program is to be run.

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 run. The term is synonymous with execution time.

OBJECT-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the object program is run, is described.

Obsolete Element. A COBOL language element in ANSI COBOL that is to be deleted from the next revision of ANSI COBOL.

Open Mode. The state of a file after running an OPEN statement for that file and before running a CLOSE statement without the REEL or UNIT phrase for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

Operand. Whereas the general definition of operand is 'that component which is operated upon', for the purposes of this document, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

Operational Sign. An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Optional File. A file which is declared as being not necessarily present each time the object program is run. The object program causes an interrogation for the presence or absence of the file.

Optional Word. A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Output File. A file that is opened in either the output mode or extend mode.

Output Mode. The state of a file after running an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before running a CLOSE statement without the REEL or UNIT phrase for that file.

Output Procedure. A set of statements to which control is given during the running of a SORT statement after the sort function is completed, or during the running of a MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

Padding Character. An alphanumeric character used to fill the unused character positions in a physical record.

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 as follows:

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
REMARKS.
```

The permissible paragraph headers in the Environment Division are as follows:

```
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.

Pattern. The object of a LIKE relation condition that specifies the regular expression used for testing the subject for a match.

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 Page. A device dependent concept defined by the action taken by a printer when a new page is requested.

Physical Record. The term is synonymous with block.

Pointer Data Item. A data item in which the address of another data item may be stored in a machine dependent form. The area of memory addressed by a pointer data item can be accessed by setting the base address of a based linkage record to the value of the pointer in a Format 5 SET statement.

Previous Record. The record which logically precedes the current record of a file.

Prime Record Key. The primary record key for an indexed file specified by the RECORD KEY clause of the file control entry. Except when the DUPLICATES phrase is specified, the contents of the prime record key uniquely identifies a record within an indexed file. The prime record key is the default key when the KEY phrase is omitted in a Format 2 READ statement or in a START statement.

Procedure. A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure Branching Statement. A statement that causes the explicit transfer of control to a statement other than the next operable statement in the sequence in which the statements are written in the source program. The procedure branching statements are: ALTER, CALL, CALL PROGRAM, EXIT, EXIT PROGRAM, GOBACK, GO TO, MERGE (with the OUTPUT PROCEDURE phrase), PERFORM and SORT (with the INPUT PROCEDURE or OUTPUT PROCEDURE phrase).

Procedure-Name. A user-defined word which 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.

Program Identification Entry. An entry in the PROGRAM-ID paragraph of the Identification Division which contains clauses that specify the program-name and assign selected program attributes to the program.

Program-Name. In the Identification Division and the end program header, a user-defined word that identifies a COBOL source program.

Pseudo-Text. A sequence of text words, comment lines, or the separator space in a source program or COBOL library bounded by, but not including, pseudo-text delimiters.

Pseudo-Text Delimiter. Two adjacent equal sign (=) characters used to delimit pseudo-text.

Punctuation Character. A character that belongs to the following set:

Character	Meaning
,	Comma
;	Semicolon
:	Colon
.	Period (full stop)
“	Quotation mark
’	Apostrophe
(	Left parenthesis
)	Right parenthesis
	Space
=	Equal sign

Qualified Data-Name. An identifier that is composed of a data-name followed by one or more set 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 screen-name which is used in a reference together with another screen-name which is the name of an item that is subordinate to the qualifier.

3. A section-name which is used in a reference together with a paragraph-name specified in that section.

4. A library-name which is used in a reference together with a text-name associated with that library.

5. A file-name which is used in a reference together with the special register LINAGE-COUNTER associated with that file.

Queue. A logical collection of messages awaiting transmission or processing.

Queue Name. A symbolic name that indicates to the message control system the logical path by which a message or a portion of a completed message may be accessible in a queue.

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. (Compare with definitions for "Dynamic Access" and "Sequential Access.")

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 item. The term is synonymous with logical record unless otherwise qualified (as in physical 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 Delimiting Technique. The method of determining the length of a record on the external storage medium for a sequential file.

Record Description. The total set of data description entries associated with a particular record. The term is synonymous with 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, a record key is either the prime record key or an alternate record key.

Record Locking Mode. The manner in which records are to be locked and unlocked within a shared input-output file. The record locking modes are automatic multiple, automatic single, manual multiple and manual single. Because automatic and manual specify how records are locked and single and multiple specify how records are unlocked, these modes are sometimes specified in the text as one of the following:

- Automatic record locking modes: automatic single and automatic multiple
- Manual record locking modes: manual single and manual multiple
- Multiple record locking modes: automatic multiple and manual multiple
- Single record locking modes: automatic single and manual single

Record Number. The ordinal number of a record in the file whose organization is sequential.

Record-Name. A user-defined word that names a record described in a record description entry in the Data Division of a COBOL program.

Reel. A discrete portion of a storage medium, the dimensions of which are determined by the physical medium, that contains part of a file, all of a file, or any number of files. The term is synonymous with unit and volume.

Reference Modifier. The farthest left-character-position and length used to establish and refer to a data item.

Regular Expression. A simple form of an expression that uses meta-characters as operators to define a pattern. The regular expressions used in RM/COBOL LIKE conditions match regulars expressions defined by XML Schema. Regular expressions are explained beginning on page 5-29. A summary of regular expression grammar is given on page 5-36.

Relation. The term is synonymous with 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.

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 as follows:

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to
IS GREATER THAN OR EQUAL TO IS >=	Greater than or equal to
IS 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 logical ordinal position of the record in the file. (Compare with definitions for "Indexed Organization" and "Sequential Organization.")

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 which may be used in a COBOL source program, but which must not appear in the program as user-defined words or system-names. (See Appendix A, *Reserved Words*, for a list of reserved words.)

Resource. A facility or service, controlled by the operating system, that can be used by a running program.

Resultant Identifier. A user-defined data item that is to contain the result of an arithmetic operation.

Routine-Name. A user-defined word that identifies a procedure written in a language other than COBOL.

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.

Screen Clause. A clause, specified in a screen description entry in the Screen Section of the Data Division of a COBOL program, that provides information describing a particular attribute of a screen item.

Screen Description Entry. An entry, in the Screen Section of the Data Division of a COBOL program, that is composed of a level-number followed by a screen-name, if required, and then followed by a set of screen clauses, as required.

Screen Item. A unit of data, including its associated screen attributes, defined by the COBOL program in the Screen Section of the Data Division.

Screen Section. The section of the Data Division that describes screen items, composed of screen records.

Screen-Name. A user-defined word that names a screen item described in a screen description entry.

Section. A set of zero, one, or more paragraphs or entries, 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 as follows:

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

The permissible section headers in the Data Division are as follows:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
COMMUNICATION SECTION.
SCREEN SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a separator period.

Section-Name. A user-defined word which names a section in the Procedure Division.

Segment-Number. A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters '0', '1', . . ., '9'. A segment-number may be expressed either as a one or two digit number.

Sentence. A sequence of one or more statements, the last of which is ended by a separator period.

Separately Compiled Program. A program which, together with its contained programs, is compiled separately from all other programs.

Separator. A character or two adjacent characters used to delimit character-strings.

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. (Compare with definitions for "Dynamic Access" and "Random Access.")

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. (Compare with definitions for "Indexed Organization" and "Relative Organization.")

Shared File. A file that is open with a lock mode of shared.

Shared File Environment. An execution environment for COBOL programs in which concurrently executing run units may asynchronously access the same physical files. Examples of such environments are multitasking operating systems and network file systems (such as local area networks which are often called LANs).

Shared Mode. A lock mode in which the file may be in an open mode concurrently by more than one run unit. When shared input-output mode applies, record locking also applies in order to coordinate access to and updating of individual records. A file cannot be successfully opened in shared mode if any other run unit has the file open in a conflicting exclusive mode.

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)
```

Single Record Locking Modes. Record locking modes in which locked records are to be unlocked implicitly by the execution of any input-output statement that refers to the file. Thus, only at most a single record at a time is locked in the file by the run unit. The single record locking modes are automatic single and manual single.

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 identification of the originator of a transmission to a queue.

Source Computer Entry. An entry in the SOURCE-COMPUTER paragraph of the Environment Division which contains clauses which describe the computer environment in which the source program is to be compiled.

Source Format. A format that provides a standard method for describing COBOL source programs.

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; a COPY statement; or a REPLACE statement. A COBOL source program is ended by the end program header, if specified, or by the absence of additional source program lines.

SOURCE-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

Special Character. A character that belongs to the following set:

Character	Meaning
+	Plus sign
–	Minus sign
*	Asterisk
/	Slant (solidus)
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point, full stop)
“	Quotation mark
’	Apostrophe
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol
:	Colon

Special Character Word. A reserved word which is an arithmetic operator or a relation character.

Special Names Entry. An entry in the SPECIAL-NAMES paragraph of the Environment Division which provides means for specifying the currency sign; choosing the decimal point; specifying symbolic characters; relating feature-names, switch-names, and low-volume-I-O-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters.

Special Registers. Certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features.

SPECIAL-NAMES. The name of an Environment Division paragraph which provides means for specifying the currency sign; choosing the decimal point; specifying symbolic characters; relating feature-names, switch-names, and low-volume-I-O-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters.

Split Key. A record key of an indexed file that is the concatenation of one or more data items with a record associated with the file. The data items need not be contiguous within the record. The split key is specified in READ and START statements with a split-key-name.

Split-Key-Name. A user-defined word that names a concatenation of one or more data items within a record associated with an indexed file. The concatenation of the data items forms a single record key for that file. A split-key-name may be specified only in a READ or START statement.

Standard Data Format. The concept used in describing 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 medium.

Statement. A syntactically valid combination of words, literals, and separators, beginning with a verb, written in a COBOL source program.

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. A program which 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 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-Name. A system-name that names a switch in the operating environment.

Switch-Status Condition. The proposition, for which a truth value can be determined that a switch, capable of being set to an ‘on’ or ‘off’ status, has been set to a specific status.

Symbolic-Character. A user-defined word that specifies a user-defined figurative constant.

System-Name. A COBOL word which is used to communicate with the operating environment.

Table. A set of logically consecutive items of data that are defined in the Data Division of a COBOL program by means of the OCCURS clause.

Table Element. A data item that belongs to the set of repeated items comprising a table.

Table-Name. A data-name that includes the OCCURS clause in its data description entry.

Terminal. The originator of a transmission to a queue or the receiver of a transmission from a queue.

Text Word. A character or a sequence of adjacent characters between margin A and margin R in a COBOL library, source program, or in pseudo-text which is:

1. 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.

2. A literal including, in the case of nonnumeric literals, the opening quotation mark and the closing quotation mark which bound the literal.

3. Any other sequence of adjacent COBOL characters except comment lines and the word 'COPY', bounded by separators, which is neither a separator nor a literal.

Text-Name. A user-defined word which identifies library text.

Top Margin. An empty area which precedes the page body.

Truth Value. The representation of the result of the evaluation of a condition in terms of one of two values: true, false.

Unary Operator. A plus (+) or a minus (–) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or –1 respectively.

Unicode. Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. Information about Unicode is available at http://www.unicode.org.

Unit. A discrete portion of a storage medium, the dimensions of which are determined by the physical medium, that contains part of a file, all of a file, or any number of files. The term is synonymous with reel and volume.

Unsuccessful Execution. The attempted running of a statement that does not result in the running of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referred by that statement, but may affect status indicators.

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-Length Record. A record associated with a file whose file description or sort-merge description entry permits records to contain a varying number of character positions.

Variable-Occurrence Data Item. A variable-occurrence data item is a table element which is repeated a variable number of times. Such an item must contain an OCCURS DEPENDING ON clause in its data description entry, or be subordinate to such an item.

Verb. A word that expresses an action to be taken by a COBOL compiler or object program.

Volume. A discrete portion of a storage medium, the dimensions of which are determined by the physical medium, that contains part of a file, all of a file, or any number of files. The term is synonymous with reel and unit.

Word. A character-string of not more than 30 characters which 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 of both.

XML. The abbreviation denoting the eXtensible Markup Language, a language for specifying documents. The specification for XML is available at http://www.w3.org/TR/2000/REC-xml-20001006.

XML Schema. The specification that describes a definition language for XML documents. The specification is available at http://www.w3.org/XML/Schema.

Zero-Length Item. An item whose minimum permitted length is zero and whose length at execution time is zero.

Index

6

66-level-description-entry, 1-7, 4-31
  format, 4-26, 4-47
  glossary term, C-1

7

77-level-description-entry, 1-7, 4-31
  format, 4-24
  glossary term, C-1
78-level-description-entry, 1-6, 4-31
  format, 4-26
  glossary term, C-1

8

88-level-description-entry, 1-7, 4-31
  format, 4-26
  glossary term, C-1

A

Abbreviated combined relation condition, 5-42
  glossary term, C-1
ACCEPT . . . FROM statement, 6-2
ACCEPT MESSAGE COUNT statement, 6-24
ACCEPT Screen-Name statement, 6-25
ACCEPT statement (terminal I-O), 6-6
Access mode. *See also* Dynamic access, Random access, and Sequential access
  for indexed file organization, 5-60
  for relative file organization, 5-51
  for sequential file organization, 5-44
  glossary term, C-1
ACCESS MODE clause, file control entry, 3-23
Actual argument
  CALL statement, 4-21, 5-3, 6-36
  glossary term, C-1
  Linkage Section, 1-11, 4-1, 4-19, 4-21
Actual decimal point
  editing symbol ".", 4-39
  glossary term, C-1
ADD statement, 5-16, 6-29
ADDRESS phrase, SET statement, 6-178
ADDRESS special register, 1-11, 4-19, 5-4, 5-25, 6-36, 6-38
ADVANCING phrase
  SEND statement, 6-176
  WRITE statement, 6-215
AFTER phrase
  INSPECT statement, 6-101
  PERFORM statement, 6-132
  SEND statement, 6-177
  WRITE statement, 6-216
ALL literal, figurative constant, 1-17
ALL phrase
  INSPECT REPLACING statement, 6-102
  INSPECT TALLYING statement, 6-102
  UNSTRING statement, 6-208
ALL TO VALUE phrase, INITIALIZE statement, 6-93
ALPHABET clause, 3-10
  code name alphabets, 3-13
  literal alphabets, 3-13
Alphabetic character
  alphabetic data item, 4-35
  class condition, 5-38
  CONTROL phrase, ACCEPT statement, 6-10
  CURRENCY SIGN clause, 3-17
  glossary term, C-1
  PICTURE character-string, 4-36
  user-defined words, 1-4, 1-35
Alphabetic class, 4-95

ALPHABETIC class condition, conditional expressions, 5-37
Alphabetic data item, 4-35
ALPHABETIC phrase, INITIALIZE statement, 6-92
ALPHABETIC-LOWER class condition, conditional expressions, 5-37
ALPHABETIC-UPPER class condition, conditional expressions, 5-37
Alphabet-name
  ALPHABET clause, 3-10
  CODE-SET clause, 3-26, 4-8
  glossary term, C-2
  MERGE statement, 6-106
  PROGRAM COLLATING SEQUENCE clause, 3-6
  scope, 1-31
  SORT statement, 6-183
  SYMBOLIC CHARACTERS clause, 3-15
  user-defined word type, 1-5
Alphanumeric character
  glossary term, C-2
  regular expressions, Perl language, 5-31
Alphanumeric class, 4-95
Alphanumeric data item, 4-35, 5-28
Alphanumeric edited data item, 4-36
ALPHANUMERIC phrase, INITIALIZE statement, 6-92
ALPHANUMERIC-EDITED phrase, INITIALIZE statement, 6-92
ALSO phrase, EVALUATE statement, 6-79
ALTER statement, 6-32
  restrictions, 5-11
ALTERNATE phrase, RESERVE clause, file control entry, 3-22, 3-34
Alternate record key, 3-32, 5-59
  glossary term, C-2
  NEXT phrase, Format 1 READ statement, 6-150
  REWRITE statement, 6-165
  WRITE statement, 6-218
ALTERNATE RECORD KEY clause, file control entry, 3-32, 5-59
American Standard Code for Information Interchange. *See* ASCII
AND logical operator
  abbreviated combined relation conditions, 5-42
  combined conditions, 5-41
  constant-expressions, 4-63
ANY, selection object, 6-81
Area A, illustrated and rules defined, 1-18
Area B, illustrated and rules defined, 1-18
Argument, actual
  CALL statement, 4-21, 5-3, 6-36
  glossary term, C-1
  Linkage Section, 1-11, 4-1, 4-19, 4-21
Argument, formal
  glossary term, C-13
  Linkage Section, 4-1, 4-19, 4-103
  procedure division header, 5-3
Arithmetic expressions, 5-20
  COMPUTE statement, 6-48
  EVALUATE statement, 6-79
  glossary term, C-2
  parentheses, using in, 1-2, 5-20, 5-22
  reference modification, 4-102
  relate condition, 5-24
  SEARCH statement (ALL), 6-167
  Sign condition, 5-39
Arithmetic operation, 5-16–5-19, 5-23
  glossary term, C-2
Arithmetic operators, 4-62, 5-22
  glossary term, C-2
Arithmetic statements, 1-21, 5-16
  ADD statement, 6-29
  COMP usage, 4-54–4-55
  COMPUTE statement, 6-48
  DIVIDE statement, 6-71
  glossary term, C-2
  MULTIPLY statement, 6-119
  subscripting, 4-101
  SUBTRACT statement, 6-203

Ascending key
glossary term, C-2
OCCURS clause, KEY phrase, 4-34
ASCENDING phrase
MERGE statement, 6-108
OCCURS clause, 4-32
SORT statement, 6-185
ASCII
STANDARD-1 alphabet, 3-10, 3-13
translation, 3-15
ASSIGN clause
file control entry, 3-25
sort-merge file control entry, 3-35
Assumed decimal point
data alignment, 4-96
glossary term, C-2
PICTURE character-string symbol "V", 4-38
At end condition
glossary term, C-3
indexed file, 5-62, 5-67
relative file, 5-53, 5-57
sequential file, 5-46, 5-49
AT END phrase
indexed file, 5-67
READ statement, 6-149
relative file, 5-57
RETURN statement, 6-161
SEARCH statement (ALL), 6-169
SEARCH statement (serial), 6-168
sequential file, 5-49
AUTHOR paragraph, 2-3
AUTO clause, screen description entry, 4-84
Automatic multiple, 5-71
glossary term, C-3
AUTOMATIC phrase, LOCK MODE clause, file control entry, 3-28
Automatic record locking modes, 3-28, 5-71
glossary term, C-3
Automatic single, 3-28, 5-71
glossary term, C-3

B

BACKGROUND clause, screen description entry, 4-84
BACKGROUND-COLOR clause, screen description entry, 4-84
Base address
based linkage, 4-19
SET statement, 6-178
Based linkage record, 4-19
glossary term, C-3
BEEP clause, screen description entry, 4-85
BEEP phrase
ACCEPT statement, 6-9
DISPLAY statement, 6-62
BEFORE phrase
INSPECT statement, 6-101
SEND statement, 6-176
WRITE statement, 6-216
BEFORE TIME phrase, ACCEPT statement, 6-21
BELL clause, screen description entry, 4-85
BELL phrase, ACCEPT statement, 6-9
Binary allocation override, 4-52, 4-55, 5-2
glossary term, C-3
parentheses, using in, 1-2, 4-55
Binary sequential
glossary term, C-4
record delimiting technique, 3-29, 3-31
BINARY usage, data description entry, 4-55
BINARY-ALLOCATION keyword, COMPILER-OPTIONS configuration record, 4-55, 5-2
BINARY-ALLOCATION-SIGNED keyword, COMPILER-OPTIONS configuration record, 4-55, 5-2
BINARY-SEQUENTIAL record delimiting technique
nonreserved system-names, A-10
RECORD DELIMITER clause, 3-31
system-names, 1-10
BLANK LINE clause, screen description entry, 4-85
Blank lines, source format, 1-19

BLANK REMAINDER clause, screen
description entry, 4-86
BLANK SCREEN clause, screen description
entry, 4-86
BLANK WHEN ZERO clause
data description entry, 4-28
screen description entry, 4-86
BLINK clause, screen description entry,
4-87
BLINK phrase
ACCEPT statement, 6-10
DISPLAY statement, 6-62
Block
BLOCK CONTAINS clause, 4-7
glossary term, C-4
BLOCK CONTAINS clause, file description
entry, 4-7
Bottom margin, 4-11, 4-14
glossary term, C-4
BY CONTENT phrase, CALL statement,
6-37
BY phrase
COPY statement, 1-35, 1-39
INSPECT statement, 6-102
PERFORM statement, 6-133
REPLACE statement, 1-40
SET statement, 6-178
BY REFERENCE phrase, CALL statement,
6-37

C

C$CARG subprogram, 4-21
C$CompilePattern subprogram, 5-28
C$MemoryAllocate subprogram, 4-57
C01–C12 channel-names
nonreserved system-names, A-9
SPECIAL-NAMES paragraph, 3-9
system-names, 1-9
CALL PROGRAM statement, 6-41
CALL statement, 6-33
Called program
CALL statement, 6-34
glossary term, C-4
Calling program
CALL statement, 6-34
glossary term, C-4
CANCEL statement, 6-43
CARD-PUNCH device-name
ASSIGN clause, 3-25
nonreserved system-names, A-10
system-names, 1-9
CARD-READER device-name
ASSIGN clause, 3-25
nonreserved system-names, A-10
system-names, 1-9
CASE-INSENSITIVE phrase, LIKE relation
condition, 5-28
CASE-SENSITIVE phrase, LIKE relation
condition, 5-28
CASSETTE device-name
ASSIGN clause, 3-25
nonreserved system-names, A-10
system-names, 1-9
CD. *See* Communication description entry
Cd-name
ACCEPT MESSAGE COUNT statement,
6-24
communication description entry, 4-64
DISABLE statement, 6-55
ENABLE statement, 6-75
glossary term, C-4
PURGE statement, 6-145
RECEIVE statement, 6-155
scope, 1-30
SEND statement, 6-173
user-defined word type, 1-5
CENTURY-DATE phrase, ACCEPT FROM
statement, 6-3
CENTURY-DAY phrase, ACCEPT FROM
statement, 6-3
Channel-name, glossary term, C-4
Channel-names, C01–C12
nonreserved system-names, A-9
SPECIAL-NAMES paragraph, 3-9
system-names, 1-9

Character, 1-1
  glossary term, C-4
Character code set, ALPHABET clause, 3-10
Character position, glossary term, C-4
Character set, 1-1
  list of, 1-2
CHARACTERS phrase
  INSPECT REPLACING statement, 6-102
  INSPECT TALLYING statement, 6-102
Characters, special, 1-13
Character-strings, 1-4, 4-34
  COBOL words
    length of, 1-4
    system-names, list of, 1-9
    user-defined, 1-4, 1-5–1-8
  comment-entry, 1-17
  glossary term, C-4
  literals, figurative constants, rules for determining string length, 1-16
  PICTURE, 1-17
    in PICTURE clause (data description entry), 4-34
    in PICTURE clause (screen description entry), 4-92
  separators, 1-1
CLASS clause, 3-16
Class condition, 3-16, 5-37
  glossary term, C-4
Classes of data, 4-95
Class-name
  CLASS clause, 3-16
  class condition, 5-37
  glossary term, C-4
  scope, 1-31
  user-defined word type, 1-5
Clause
  glossary term, C-4
  program structure, 1-24
CLOSE statement, 6-45
  implicit CLOSE on CANCEL, 6-44
COBOL character set, 1-2
  glossary term, C-5
COBOL words
  context-sensitive, 1-4, 1-14, A-7
  disjoint sets, 1-4
  glossary term, C-5
  reserved, 1-4, 1-10, A-1–A-6
  system-names, 1-4, 1-8, A-9–A-11
  user-defined, 1-4, 1-5–1-8
Code-name
  ALPHABET clause, 3-10
  alphabets, 3-13
  EBCDIC, 1-9, A-9
  glossary term, C-5
  nonreserved system-names, A-9
  system-names, 1-8–1-9
CODE-SET clause
  file control entry, 3-26
  file description entry, 4-8
Collating sequence, 3-6, 3-27
  ALPHABET clause, 3-10
  comparison of nonnumeric operands, 5-26
  glossary term, C-5
  indexed file, 3-27
  MERGE statement, 6-108
  SORT statement, 6-185
COLLATING SEQUENCE clause
  file control entry, 3-27
  PROGRAM OBJECT-COMPUTER paragraph, 3-6
COLLATING SEQUENCE phrase
  MERGE statement, 6-108
  SORT statement, 6-185
Colons, using as separators, 1-3
Color-name
  BACKGROUND clause, 4-84
  FOREGROUND clause, 4-88
  nonreserved system-names, A-11
Column
  COLUMN (COL) phrase
    ACCEPT statement (terminal I-O), 6-17
    DISPLAY statement (terminal I-O), 6-65

COLUMN clause, screen description entry, 4-87
glossary term, C-5
LINE and POSITION phrases
ACCEPT statement (terminal I-O), 6-17
DISPLAY statement (terminal I-O), 6-65
COLUMN (COL) phrase
ACCEPT Screen-Name statement, 6-25
ACCEPT statement (terminal I-O), 6-17
DISPLAY Screen-Name statement, 6-69
DISPLAY statement (terminal I-O), 6-65
COLUMN clause, screen description entry, 4-87
Combined condition, 5-41
glossary term, C-6
Comma
DECIMAL-POINT clause, SPECIAL-NAMES paragraph, 3-18
numeric literals, 1-14
PICTURE character-string, 4-39
using as separators, 1-2
Comment lines, 1-1, 1-19
glossary term, C-6
source format, 1-19
with asterisk, 1-19
Comment-entry, 1-17, 2-1
AUTHOR paragraph, 2-3
DATE-COMPILED paragraph, 2-3
DATE-WRITTEN paragraph, 2-3
glossary term, C-6
INSTALLATION paragraph, 2-3
REMARKS paragraph, 2-3
SECURITY paragraph, 2-3
Comments, 1-18
in-line, 1-20
COMMON clause, PROGRAM-ID paragraph, 2-2
Common program, 1-29, 2-2
glossary term, C-6
Common rules, procedure division, 5-16
Communication description entry, 4-64
glossary term, C-6
Communication device
glossary term, C-6
relationship to Message Control System, object program, 5-76
Communication facility, 5-75
Communication Section
Data Division, 4-2, 4-22
glossary term, C-6
Communication statements
DISABLE, 6-55
ENABLE, 6-75
PURGE, 6-145
RECEIVE, 6-155
SEND, 6-173
Comparison
index-names and index data items, 5-27
nonnumeric operands, 5-26
numeric operands, 5-25
pointer data items, 5-27
Compile time
glossary term, C-6
Compiler directing statement
COPY, 1-35
ENTER, 6-78
glossary term, C-6
REPLACE, 1-40
USE, 5-13
Compiler, messages, B-1
COMPILER-OPTIONS configuration record
BINARY-ALLOCATION keyword, 4-55, 5-2
BINARY-ALLOCATION-SIGNED keyword, 4-55, 5-2
COMPUTATIONAL-TYPE keyword, 4-52–4-53
DEFAULT-USE-PROCEDURE keyword, 5-15, 5-49–5-50, 5-56–5-58, 5-66–5-68, 6-149
DERESERVE keyword, A-6, A-8

ENTRY-LINKAGE-SETTINGS keyword, 4-20
LISTING-DATE-FORMAT keyword, 2-3
LISTING-DATE-SEPARATOR keyword, 2-3
OBJECT-VERSION keyword, B-84
SUPPRESS-FILLER-IN-SYMBOL-TABLE keyword, B-93
SUPPRESS-LITERAL-BY-CONTENT keyword, 6-37
SYMBOL-TABLE-OUTPUT keyword, B-84
Complex condition, 5-40
glossary term, C-6
Composite of operands, 5-23, 6-48, B-20
glossary term, C-6
Composite size, 5-17
COMPUTATIONAL usage, 4-53
COMPUTATIONAL-1 usage, 4-54
COMPUTATIONAL-3 usage, 4-54
COMPUTATIONAL-4 usage, 4-55
COMPUTATIONAL-6 usage, 4-56
COMPUTATIONAL-TYPE keyword, COMPILER-OPTIONS configuration record, 4-52–4-53
COMPUTE statement, 5-16, 6-48
Computer-name, 3-5, 3-6
glossary term, C-7
Condition evaluation rules, 5-43
Conditional
phrases, 1-22
glossary term, C-7
sentences, 1-24
statements, 1-21
glossary term, C-7
Conditional expressions, 5-23. *See also* Conditions
complex, 5-40
glossary term, C-7
simple, 5-23
Conditional variable
condition-name condition, 5-39
condition-name VALUE clause, 4-61
Format 3 data description entry, 4-27
glossary term, C-7
Condition-name, 1-30, 4-26, 4-58
conditional variable condition, 5-39
data description entry, 4-26
global, 1-27, 4-30
glossary term, C-7
qualification, 4-97
references, 4-103
rules (Format 2 VALUE clause), 4-61
scope, 1-32
switch-status, 3-7, 3-9
switch-status condition, 5-40
tests, 3-9
user-defined word type, 1-5
Condition-name condition
conditional expressions, 5-39
glossary term, C-7
Conditions
abbreviated combined, 5-42, C-1
class, 5-37, C-4
combined, 5-41, C-6
complex, 5-40, C-6
conditional expressions, 5-23, 5-40, C-7
condition-name, 5-39, C-7
evaluation rules, 5-43
glossary term, C-7
LIKE (relation condition), 5-27, C-16
negated, 5-41, C-19
parentheses, using in, 1-2, 5-23
relation, 5-24, C-25
sign, 5-39, C-28
simple, 5-23, C-29
switch-status, 5-40, C-31
Configuration Section, 3-5
glossary term, C-7
Connectives, 1-10

CONSOLE device-name
ASSIGN clause, 3-25
nonreserved system-names, A-9–A-10
system-names, 1-9
CONSOLE low-volume-I-O-name, 3-9
ACCEPT statement, 6-2
DISPLAY statement, 6-58
nonreserved system-names, A-9
system-names, 1-10
Constant-expressions
DATE-COMPILED phrase, 4-64
format, 4-62
glossary term, C-8
logical operators, 4-63
parentheses, using in, 1-2, 4-62
rules, 4-62
VALUE clause, 4-58, 4-62
Constant-name, 4-26, 4-58
data description entry, 4-26
global, 1-28
glossary term, C-8
rules (Format 3 VALUE clause), 4-62
scope, 1-32
user-defined word type, 1-6
Contained program
directly, 1-27
indirectly, 1-27
Context-sensitive words, 1-4, 1-14, A-7
glossary term, C-8
Contiguous items
glossary term, C-8
record description entry, 4-23
Continuation line, 1-19
CONTINUE statement, 6-50
Continued line, source format, 1-19
CONTROL phrase
ACCEPT statement, 6-8, 6-10
DISPLAY statement, 6-61, 6-62
Conventions and symbols, xx
CONVERT phrase
ACCEPT statement, 6-11
DISPLAY statement, 6-63
CONVERTING phrase, INSPECT statement, 6-97
COPY statement, 1-35
BY phrase, 1-35, 1-39
REPLACING phrase, 1-35, 1-37
CORRESPONDING phrase
ADD statement, 6-30
MOVE statement, 6-116
SUBTRACT statement, 6-204
COUNT clause, input CD entry, 4-68
COUNT phrase, UNSTRING statement, 6-209
COUNT special register, 1-11
Counter, glossary term, C-8
COUNT-MAX special register, 1-11
COUNT-MIN special register, 1-12
Critical error conditions
indexed, 5-61
relative, 5-51
sequential, 5-44
cs. *See* Currency sign
Currency sign. *See also* Currency symbol
CURRENCY SIGN clause, SPECIAL-NAMES paragraph, 3-17
glossary term, C-8
PICTURE character-string, 4-40
CURRENCY SIGN clause, 3-17
PICTURE character-string, 4-40
SPECIAL-NAMES paragraph, 3-17
Currency symbol. *See also* Currency sign
CURRENCY-SIGN clause, SPECIAL-NAMES paragraph, 3-17
glossary term, C-8
PICTURE character-string, 4-40
Current record
glossary term, C-8
READ statement, 6-146
Current volume pointer
CLOSE statement, REEL and UNIT phrases, 6-46
glossary term, C-8

CURSOR phrase, ACCEPT statement, 6-13
CYCLE phrase, EXIT PERFORM
statement, 6-85

D

Data
external, 4-29
structure, classes of, 4-95
DATA BY phrase, INITIALIZE statement, 6-92
Data clause
BLANK WHEN ZERO, 4-28
data-name or FILLER, 4-29
EXTERNAL, 4-29
GLOBAL, 4-30
glossary term, C-8
JUSTIFIED, 4-30
level-number, 4-31
OCCURS, 4-32
PICTURE, 4-34
REDEFINES, 4-46
RENAMES, 4-26, 4-47
SIGN, 4-48
SYNCHRONIZED, 4-50
USAGE, 4-52
VALUE, 4-58
Data description entry, 4-24
BLANK WHEN ZERO clause, 4-28
condition-name declaration, 4-26
constant-name declaration, 4-26
data-name or FILLER clause, 4-29
EXTERNAL clause, 4-29
GLOBAL clause, 4-30
glossary term, C-9
JUSTIFIED clause, 4-30
level-number, 4-31
OCCURS clause, 1-11, 4-24, 4-32
PICTURE clause, 4-34
REDEFINES clause, 4-46
RENAMES clause, 4-26, 4-47
rules
condition-name, 4-61
constant-name, 4-62
VALUE clause, 4-60
SIGN clause, 4-48
SYNCHRONIZED clause, 4-50
USAGE clause, 4-52
VALUE clause, 4-58
Data Division, 4-1
77-level description entry, 4-24
communication description entry, 4-64
Communication Section, 4-2, 4-22
data description entry, 4-25
data structures, 4-95
classes of data, 4-95
standard alignment rules, 4-96
file description clauses, 4-7
file description entry, 4-5
File Section, 4-1, 4-4
header, 4-3
identifier, 4-103
Linkage Section, 4-1, 4-19
record description entry, 4-23
reference modification, 4-102, 5-19
screen description entry, 4-80
screen field format, 4-82
screen group format, 4-80
screen literal format, 4-81
Screen Section, 4-2, 4-22
sort-merge file description entry, 4-6
subscripting, 4-100
table handling, 4-104
references to table items, 4-106
table definition, 4-104
uniqueness of reference, 4-97
qualification, 4-97
Working-Storage Section, 4-1, 4-18
Data item
data pointer, 4-36, 4-57, 4-59, 6-92, 6-93
external, 4-29
file status, 5-44, 5-51, 5-60
glossary term, C-9
internal, 1-28
variable-occurrence, 4-33
DATA phrase, RECEIVE statement, 6-156
Data pointer, 4-36, 4-57, 4-59, 6-92, 6-93

DATA RECORDS clause, file description entry, 4-8
Data-name
  global, 1-27, 4-9, 4-30
  glossary term, C-9
  or FILLER clause, 4-29
  scope, 1-32
  user-defined word type, 1-6
DATA-POINTER phrase, INITIALIZE statement, 6-93
Date
  ACCEPT FROM CENTURY-DATE statement, 6-3
  ACCEPT FROM CENTURY-DAY statement, 6-3
  ACCEPT FROM DATE statement, 6-3
  ACCEPT FROM DATE-AND-TIME statement, 6-4
  ACCEPT FROM DATE-COMPILED statement, 6-4
  listing format configuration, 2-3
  listing separator configuration, 2-3
DATE phrase, ACCEPT FROM statement, 6-3
DATE-AND-TIME phrase, ACCEPT FROM statement, 6-4
DATE-COMPILED paragraph, 2-3
DATE-COMPILED phrase
  ACCEPT FROM statement, 6-4
  constant-expressions, 4-64
DATE-WRITTEN paragraph, 2-3
Day
  ACCEPT FROM DAY statement, 6-4
  ACCEPT FROM DAY-AND-TIME statement, 6-4
  ACCEPT FROM DAY-OF-WEEK statement, 6-4
DAY phrase, ACCEPT FROM statement, 6-4
DAY-AND-TIME phrase, ACCEPT FROM statement, 6-4
DAY-OF-WEEK phrase, ACCEPT FROM statement, 6-4
Debugging lines
  DEBUGGING MODE clause, SOURCE-COMPUTER paragraph, 3-5
  glossary term, C-9
  source format, 1-18, 1-20, 1-38
DEBUGGING MODE clause, SOURCE-COMPUTER paragraph, 3-5
Decimal point
  DECIMAL-POINT clause, SPECIAL-NAMES paragraph, 1-14, 3-18
  numeric literals, 1-14
  PICTURE character-string, 4-39
DECIMAL-POINT clause, SPECIAL-NAMES paragraph, 3-18
  ACCEPT statement
    input conversion, 6-12
    output conversion, 6-22
  DISPLAY statement
    output conversion, 6-63
  numeric literals, 1-14
  PICTURE character-string, 4-39
Declarative sentence
  DECLARATIVES, Procedure Division format, 5-4
  glossary term, C-9
  USE statement, 5-13
Declaratives
  glossary term, C-9
  Procedure Division format, 5-4
De-edit
  glossary term, C-9
  MOVE statement, 6-115
DEFAULT phrase, INITIALIZE statement, 6-94
DEFAULT-USE-PROCEDURE keyword, COMPILER-OPTIONS configuration record, 5-15, 5-49–5-50, 5-56–5-58, 5-66–5-68, 6-149
DELETE FILE statement, 6-54
DELETE statement (relative and indexed I-O), 6-51

DELIMITED phrase
STRING statement, 6-200
UNSTRING statement, 6-208
Delimited scope statement, 1-21–1-24
glossary term, C-9
Delimiter
glossary term, C-9
pseudo-text, 1-3
DELIMITER phrase, UNSTRING statement, 6-209
DEPENDING ON phrase
GO TO statement, 6-89
RECORD VARYING clause, 4-15
DERESERVE keyword, COMPILER-OPTIONS configuration record, A-6, A-8
Descending key
glossary term, C-9
OCCURS clause, KEY phrase, 4-34
DESCENDING phrase
MERGE statement, 6-108
OCCURS clause, 4-32
SORT statement, 6-185
Destination
glossary term, C-9
output communication description entry, 4-72
DESTINATION clause, output CD entry, 4-72
DESTINATION COUNT clause, output CD entry, 4-72
Determining line and position, 6-18, 6-66
Determining the method of scheduling, 5-78
Device-name
ASSIGN clause
file control entry, 3-25
sort-merge file control entry, 3-35
glossary term, C-9
nonreserved system-names, A-10
system-names, 1-9
Digit position, glossary term, C-9
Directive sentences, 1-24
Directive statements, 1-21
Directly contained program, 1-27
DISABLE statement, 6-55
DISC device-name
ASSIGN clause, 3-25
nonreserved system-names, A-10
system-names, 1-9
Disjoint sets, 1-4
DISK device-name
ASSIGN clause, 3-25
nonreserved system-names, A-10
system-names, 1-9
DISPLAY . . . UPON statement, 6-58
DISPLAY device-name
ASSIGN clause, 3-25
system-names, 1-9
DISPLAY Screen-Name statement, 6-69
DISPLAY statement (terminal I-O), 6-60
DISPLAY usage
data description entry, 4-56
screen description entry, 4-94
DIVIDE statement, 5-16, 6-71
Division header
Data Division, 4-3
Environment Division, 3-2
glossary term, C-10
Identification Division, 2-1
Procedure Division, 5-1
Divisions, 1-25. *See also* Division header
Data, 1-25, 4-1
Environment, 1-25, 3-1
glossary term, C-10
Identification, 1-25, 2-1
Procedure, 1-25, 5-1
DOWN BY phrase, SET statement, 6-178
DUPLICATES phrase
ALTERNATE RECORD KEY clause, file control entry, 3-33
DELETE statement, 6-51
I-O status value 43, 5-64
RECORD KEY clause, file control entry, 3-32
REWRITE statement, 6-163, 6-164
SORT statement, 6-185
WRITE statement, 6-214, 6-218

Dynamic access
glossary term, C-10
mode
for indexed file organization, 5-60
for relative file organization, 5-51

E

EBCDIC code-name, 3-11, 3-13
nonreserved system-names, A-9
system-names, 1-9
translation, 3-15
ECHO phrase, ACCEPT statement, 6-13
Editing (PICTURE clause)
fixed insertion, 4-41
floating insertion, 4-42
general rules, 4-40
simple insertion, 4-41
special insertion, 4-41
zero suppression, 4-43
Editing characters, 4-36
glossary term, C-10
Elementary items, 4-24
glossary term, C-10
ELSE phrase, IF statement, 6-90
ENABLE statement, 6-75
Enabling and disabling queues, 5-81
END KEY clause
input CD entry, 4-68
input-output CD entry, 4-74
End of procedure division, glossary term, C-10
End program header, 1-12, 1-25, 1-34
glossary term, C-10
END-OF-PAGE phrase, WRITE statement, 6-217
ENTER statement, 6-78
Entry
glossary term, C-11
program structure, 1-24
ENTRY-LINKAGE-SETTINGS keyword, COMPILER-OPTIONS configuration record, 4-20
Environment clause, glossary term, C-11
Environment Division, 3-1
Configuration Section, 3-5
OBJECT-COMPUTER paragraph, 3-6
SOURCE-COMPUTER paragraph, 3-5
SPECIAL-NAMES paragraph, 3-7
Input-Output Section, 3-18
FILE-CONTROL paragraph, 3-19
I-O-CONTROL paragraph, 3-36
EOP phrase. *See* END-OF-PAGE phrase
EQUAL relation condition, conditional expressions, 5-24
ERASE clause, screen description entry, 4-88
ERASE phrase
ACCEPT statement, 6-14
DISPLAY statement, 6-64
ERROR KEY clause, output CD entry, 4-72
Error key values, 4-79
ERROR phrase, USE statement, 5-13
Escape condition, ACCEPT Screen-Name statement, 6-27
ESCAPE KEY, ACCEPT statement, 6-4
ESCAPE phrase
ACCEPT (terminal I-O)statement, 6-14
ACCEPT Screen-Name statement, 6-27
EVALUATE statement, 5-23, 6-79
EXCEPTION phrase
ACCEPT statement, 6-14
CALL PROGRAM statement, 6-42
CALL statement, 6-39
USE statement, 5-13
EXCEPTION STATUS, ACCEPT statement, 6-4
Exclusive file
file locking, 5-68
glossary term, C-11
Exclusive mode
file locking, 3-28, 5-68
glossary term, C-11
EXCLUSIVE OR logical operator, constant-expressions, 4-63

EXCLUSIVE phrase
LOCK MODE clause, file control entry, 3-28
OPEN statement, 6-121
file locking, 5-68
Execution time, 5-6
glossary term, C-11
EXIT statement, 6-84
Explicit scope terminator
glossary term, C-11
in Procedure Division statements, 1-23
Exponentiation
arithmetic-expressions, 5-23
constant-expressions, 4-62
Expressions
arithmetic, 5-20
conditional, 5-23, 5-40
constant, 4-62
glossary term, C-11
regular, 5-29
Extend mode
glossary term, C-11
OPEN statement, 6-126
EXTEND phrase
OPEN statement, 6-121, 6-126
USE statement, 5-14
Extended Binary Coded Decimal
Interchange Code. *See* EBCDIC
code-name
EXTERNAL clause
data description entry, 4-29
file description entry, 4-9
External data, 4-29
glossary term, C-11
External data item
EXTERNAL clause, data description entry, 4-29
glossary term, C-11
External data record, glossary term, C-11
External file connector
EXTERNAL clause, file description entryv, 4-9
glossary term, C-11
External objects, 1-28
External switch
glossary term, C-11
mnemonic-names, 3-10
SET statement (ON/OFF), 6-180
switch-names, 1-10
switch-status condition, 5-40

F

FALSE phrase
SET statement, 6-178
VALUE clause, 4-58
FALSE, selection subject or object, 6-81
FD. *See* File description entry
Feature-name, 1-8
glossary term, C-12
SPECIAL-NAMES paragraph, 3-9
system-names, 1-9
Figurative constants, 1-15
glossary term, C-12
symbolic-characters, 3-15
File
file control entry, 3-20
file description entry, 4-5
glossary term, C-12
procedure division input-output
indexed, 5-59
relative, 5-50
sequential, 5-43
sort-merge file control entry, 3-35
sort-merge file description entry, 4-6
File access name
ASSIGN clause
file control entry, 3-26
sort-merge file control entry, 3-35
glossary term, C-12
VALUE OF clause, 4-17
File attribute conflict condition
DELETE FILE statement, 6-54
glossary term, C-12
I-O status 39 (indexed I-O), 5-63
I-O status 39 (relative I-O), 5-54
I-O status 39 (sequential I-O), 5-47

File availability, 6-122
File clause
  BLOCK CONTAINS, 4-7
  CODE-SET, 4-8
  DATA RECORDS, 4-8
  EXTERNAL, 4-9
  GLOBAL, 4-9
  glossary term, C-12
  LABEL, 4-9
  LINAGE, 4-10
  RECORD, 4-15
  VALUE OF, 4-17
File connector, 1-27
  external, 4-9
  glossary term, C-12
File control entry
  ACCESS MODE clause, 3-23
  ALTERNATE RECORD KEY clause, 3-32
  ASSIGN clause, 3-25
    sort-merge file control entry, 3-35
  CODE-SET clause, 3-26
  COLLATING SEQUENCE clause, 3-27
  FILE STATUS clause, 3-28
  glossary term, C-12
  LOCK MODE clause, 3-28
  ORGANIZATION clause, 3-29
  PADDING CHARACTER clause, 3-30
  RECORD DELIMITER clause, 3-31
  RECORD KEY clause, 3-32
  RESERVE clause, 3-22, 3-34
  SELECT clause, 3-21, 3-35
  sequential, relative, and indexed file organization, 3-20
  sort-merge file control entry, 3-35
File description entry, 4-5
  BLOCK CONTAINS clause, 4-7
  CODE-SET clause, 4-8
  DATA RECORDS clause, 4-8
  EXTERNAL clause, 4-9
  GLOBAL clause, 4-9
  glossary term, C-12
  LABEL RECORDS clause, 4-9
  LINAGE clause, 4-10
  RECORD clause, 4-15
  VALUE OF clause, 4-17
File locking, 5-68
  CLOSE statement, 6-47
  LOCK MODE clause, file control entry, 3-28
  OPEN statement, 6-121
File organization
  file control entry, 3-20
  glossary term, C-12
  indexed, 3-30, 5-59
  ORGANIZATION clause, file control entry, 3-29
  relative, 3-29, 5-50
  sequential, 3-29, 5-44
File position indicator
  CLOSE statement, 6-45
  glossary term, C-12
  indexed file, 5-60
  OPEN statement
    input mode, 6-124
    I-O mode, 6-125
  READ statement, 6-147
  relative file, 5-51
  sequential file, 5-44
  START statement (relative and indexed I-O), 6-193
File Section
  Data Division, 4-1, 4-4
  glossary term, C-12
FILE STATUS clause, file control entry, 3-28
File status data item
  indexed file, 5-60
  relative file, 5-51
  sequential file, 5-44
FILE-CONTROL paragraph, 3-19
  glossary term, C-13
  Input-Output Section, 3-18

FILE-ID label-name
nonreserved system-names, A-11
system-names, 1-9
VALUE OF clause, 4-17
File-name, 1-31
CLOSE statement, 6-45
DELETE FILE statement, 6-54
DELETE statement (relative and indexed I-O), 6-51
file control entry, 3-23
file description entry, 4-5–4-6
global, 1-27, 4-9
glossary term, C-13
MERGE statement, 6-106
MULTIPLE FILE TAPE clause, 3-40
OPEN statement, 6-121
qualifier, 4-97
READ statement, 6-146
RERUN clause, 3-36
RETURN statement, 6-161
SAME clause, 3-38
scope, 1-32
SORT statement, 6-183
sort-merge description entry, 4-6
START statement (relative and indexed I-O), 6-191
UNLOCK statement, 6-206
USE statement, 5-13
user-defined word type, 1-7
FILLER clause
data description entry, 4-29
screen description entry, 4-80
FILLER phrase, INITIALIZE statement, 6-93
FIRST phrase
INSPECT REPLACING statement, 6-103
START statement, 6-191
Fixed file attributes, glossary term, C-13
Fixed insertion editing, 4-41
Fixed overlayable segment, 5-9
Fixed permanent segment, 5-9
Fixed portion, 5-9
Fixed-length record, 4-16
glossary term, C-13
Floating insertion editing, 4-42
Footing area, 4-11, 4-14
glossary term, C-13
FOR REMOVAL phrase, CLOSE statement, 6-47
FORCE-USER-MODE keyword, RUN-FILES-ATTR configuration record, 5-68–5-70
FOREGROUND clause, screen description entry, 4-89
FOREGROUND-COLOR clause, screen description entry, 4-89
Formal argument
glossary term, C-13
Linkage Section, 4-1, 4-19, 4-103
procedure division header, 5-3
Format
glossary term, C-13
source, program structure, 1-18
FROM phrase
ACCEPT statement, 6-2
PERFORM statement, 6-132
PICTURE clause, screen description entry, 4-92
RELEASE statement, 6-159
REWRITE statement, 6-166
SEND statement, 6-173
SUBTRACT statement, 6-204
WRITE statement, 6-215
FULL clause, screen description entry, 4-89

G

GIVING phrase
ADD statement, 6-30
CALL statement, 6-38
DIVIDE statement, 6-72
MERGE statement, 6-110
MULTIPLY statement, 6-119
Procedure Division header, 5-3
SORT statement, 6-187
SUBTRACT statement, 6-204

Global
condition-name, 4-30
data-name, 4-30
file-name, 4-9
index-name, 1-33
GLOBAL clause
data description entry, 4-30
file description entry, 4-9
Global name
file description entry, 4-9
GLOBAL clause, data description enty, 4-30
glossary term, C-13
inter-program identification module, 1-27
GLOBAL phrase, USE statement, 5-15
GO TO statement, 6-88
GOBACK statement, 6-87
GREATER relation condition, conditional expressions, 5-24
Group item
glossary term, C-13
record description entry, 4-23
variable length, 4-33

H

High order end, 5-26
glossary term, C-13
HIGH phrase
ACCEPT statement, 6-17
DISPLAY statement, 6-64
HIGHLIGHT clause, screen description entry, 4-90
HIGHLIGHT phrase, ACCEPT statement, 6-17
HIGH-VALUE (HIGH-VALUES) figurative constant, 1-16, 3-11

I

Identification area, source format, 1-18
Identification Division
program identification, 2-1
AUTHOR, INSTALLATION, DATE-WRITTEN, SECURITY and REMARKS paragraphs, 2-3
DATE-COMPILED paragraph, 2-3
PROGRAM-ID paragraph, 2-2
Identifier
data-name, 4-103
glossary term, C-13
IF statement, 5-23, 6-90
Imperative
sentences, 1-24
statements, 1-21–1-23
verbs, list of, 1-23
Imperative statements, 1-21–1-23
glossary term, C-14
Implicit scope terminator
glossary term, C-14
in Procedure Division statements, 1-23
Incompatible data, 5-20
Independent enqueueing and dequeueing, 5-80
Independent segments, 5-10
Index data item
comparison with index-name, 5-27
data description entry, 4-56
glossary term, C-14
INDEX usage, data description entry, 4-56
Index, glossary term, C-14
INDEXED BY phrase, OCCURS clause, 4-32
defining index-names, 4-34
Indexed file
access modes, 5-60
alphabets, 3-15
file control entry, 3-20
glossary term, C-14

Indexed organization input-output, 5-59
CLOSE statement, 6-45
DELETE FILE statement, 6-54
DELETE statement, 6-51
file description entry, 4-5
indexed file control entry, 3-20
OPEN statement, 6-121
READ statement, 6-146
REWRITE statement, 6-163
START statement, 6-191
UNLOCK statement, 6-206
WRITE statement, 6-212
Indexed organization, glossary term, C-14
Index-name
comparisons, 5-27
DESTINATION TABLE clause, 4-65
glossary term, C-14
Linkage Section, 4-1
OCCURS clause, 4-32
PERFORM statement, 6-128
scope, 1-33
SEARCH statement (ALL), 6-169
SEARCH statement (serial), 6-168
SET statement, 6-178
subscripts, 4-101
user-defined word type, 1-7
Indicator area, 1-18
Indirectly contained program, 1-27
Initial attribute, 6-85
INITIAL clause
communication description entry, 4-64, 4-65
PROGRAM-ID paragraph, 2-3
Initial program, 1-29, 1-33, 2-3
glossary term, C-14
Initial state, 1-33, 4-19
glossary term, C-14
Initial state of a program
ALTER statements, 1-33
GO TO statements, 1-33
internal file connectors, 1-33
PERFORM statements, 1-33
VALUE clause, 1-33
INITIALIZE statement, 6-92
File Section, 4-4
Linkage Section, 4-20
POINTER usage, 4-57
VALUE phrase, 4-58
In-line comment
glossary term, C-14
source format, 1-20
Input code set, 3-12
INPUT device-name
ASSIGN clause, 3-25
system-names, 1-9
Input file, 3-9
glossary term, C-14
Input mode
CLOSE statement, 6-45
glossary term, C-14
OPEN statement, 6-124
INPUT phrase
DISABLE statement, 6-56
ENABLE statement, 6-76
OPEN statement, 6-124
USE statement, 5-14
Input procedure
glossary term, C-14
INPUT PROCEDURE phrase, SORT statement, 6-186
Input-output areas, RESERVE clause, 3-34
INPUT-OUTPUT device-name
ASSIGN clause, 3-25
system-names, 1-9
Input-output file
DELETE statement, 6-52
glossary term, C-14
LOCK phrase, READ statement, 6-152
REWRITE statement, 6-164
UNLOCK statement, 6-206
WRITE statement, 6-212
Input-Output Section, 3-18
glossary term, C-15

Input-output statements
ACCEPT (terminal I-O), 6-6
ACCEPT . . . FROM, 6-2
ACCEPT MESSAGE COUNT, 6-24
ACCEPT Screen-Name, 6-25
CLOSE, 6-45
DELETE (relative and indexed I-O), 6-51
DELETE FILE, 6-54
DISABLE, 6-55
DISPLAY, 6-58
DISPLAY (terminal-I-O), 6-60
DISPLAY Screen-Name, 6-69
ENABLE, 6-75
glossary term, C-15
OPEN, 6-121
PURGE, 6-145
READ, 6-146
RECEIVE, 6-155
REWRITE, 6-163
SEND, 6-173
SET . . . TO ON/OFF, 6-178
START (relative and indexed I-O), 6-191
UNLOCK, 6-206
WRITE, 6-212
Insertion editing
fixed, 4-41
floating, 4-42
simple, 4-41
special, 4-41
INSPECT statement, 6-97
INSTALLATION paragraph, 2-3
Integer, 1-14
constant-expressions, 4-62
glossary term, C-15
Interactive terminal I-O, 5-74
Internal data, 1-28
glossary term, C-15
Internal data item, 1-28
glossary term, C-15
Internal file connector, 1-28
glossary term, C-15
Internal objects, 1-28
Inter-program communication, 1-26
Intersecting sets, 1-4
INTO phrase, 4-17
READ statement, 6-153
RECEIVE statement, 6-155
RETURN statement, 6-162
STRING statement, 6-199
UNSTRING statement, 6-209
Intra-record data structure, glossary term, C-15
Invalid key condition
glossary term, C-15
indexed file, 5-62, 5-65
relative file, 5-53, 5-56
INVALID KEY phrase
DELETE statement, 6-52
indexed file, 5-65
READ statement, 6-153
relative file, 5-56
REWRITE statement, 6-165
START statement (relative and indexed I-O), 6-195
WRITE statement, 6-218
Invocation of the object program by the Message Control System, 5-78
Invoking the object program, 5-77
I-O mode
glossary term, C-15
OPEN statement, 6-125
I-O phrase
OPEN statement, 6-125
USE statement, 5-14
I-O status
FILE STATUS, file control entry, 3-28
glossary term, C-15
I-O status values
indexed file, 5-60
relative file, 5-51
sequential file, 5-44
I-O TERMINAL phrase
DISABLE statement, 6-56
ENABLE statement, 6-76

I-O-CONTROL entry
glossary term, C-16
MULTIPLE FILE TAPE clause, 3-40
RERUN clause, 3-36
SAME clause, 3-38
I-O-CONTROL paragraph
glossary term, C-15
Input-Output Section, 3-18, 3-36

J

JUSTIFIED clause
data description entry, 4-30
screen description entry, 4-90

K

Key
file control entry, 3-32
alternate, 3-32
prime, 3-32
glossary term, C-16
OCCURS clause, KEY phrase, 4-34
Key of reference
glossary term, C-16
OPEN statement
input mode, 6-124
I-O mode, 6-125
READ statement, 6-151
START statement, 6-194
KEY phrase
DISABLE statement, 6-57
ENABLE statement, 6-77
MERGE statement, 6-106
OCCURS clause, 4-32
READ statement, 6-151
SORT statement, 6-183
START statement (relative and indexed I-O), 6-192
KEYBOARD device-name
ASSIGN clause, 3-25
nonreserved system-names, A-10
system-names, 1-9
Keyword
glossary term, C-16
reserved word, 1-10

L

LABEL label-name
system-names, 1-9
VALUE OF clause, 4-17
LABEL RECORDS clause, file description entry, 4-9
Language structure, 1-1
Language-name
ENTER statement, 6-78
glossary term, C-16
LAST phrase, START statement, 6-191
LEADING phrase
INSPECT REPLACING statement, 6-102
INSPECT TALLYING statement, 6-102
SIGN clause, 4-48
LENGTH operator, constant-expressions, 4-63
LENGTH special register, 1-12, 6-182
LESS relation condition, conditional expressions, 5-24
Letters
glossary term, C-16
in character sets, 1-1
Level indicator, 4-4
CD, 4-22
FD, 4-6
glossary term, C-16
SD, 4-6
Level-number
data description entry, 4-23, 4-31
glossary term, C-16
user-defined word type, 1-7
Library text
COPY statement, 1-19, 1-35
glossary term, C-16
REPLACING phrase, COPY statement, 1-37

Library-name
COPY statement, 1-36
glossary term, C-16
scope, 1-30
user-defined word type, 1-7
LIKE relation condition
conditional expressions, 5-24, 5-27
glossary term, C-16
LINAGE clause, file description entry, 4-10
LINAGE-COUNTER
glossary term, C-16
special register, 1-12, 4-6, 4-12, 4-99
LINE clause, screen description entry, 4-90
LINE phrase
ACCEPT Screen-Name statement, 6-25
ACCEPT statement (terminal I-O), 6-17
DISPLAY Screen-Name statement, 6-69
DISPLAY statement (terminal I-O), 6-65
Line sequential
glossary term, C-17
record delimiting technique, 3-29, 3-31
Lines
blank, 1-19
comment, 1-19
continuation, 1-19
continued, 1-19
debugging, 1-18, 1-20, 1-38
in-line comment, 1-20
LINE-SEQUENTIAL record delimiting technique
nonreserved system-names, A-10
RECORD DELIMITER clause, 3-31
system-names, 1-10
Linkage Section
Data Division, 4-1, 4-19
glossary term, C-17
LISTING device-name
ASSIGN clause, 3-25
nonreserved system-names, A-10
system-names, 1-9
LISTING-DATE-FORMAT keyword, COMPILER-OPTIONS configuration record, 2-3
LISTING-DATE-SEPARATOR keyword, COMPILER-OPTIONS configuration record, 2-3
Literal alphabets, 3-13
Literals, 1-14
and figurative constants, 1-15
glossary term, C-17
nonnumeric, 1-14
numeric, 1-14
Local names, 1-27
Lock mode
file locking, 5-68
glossary term, C-17
record locking, 5-69
LOCK MODE clause
file control entry, 3-28
file locking, 5-68
record locking, 5-69
LOCK phrase
CLOSE statement, 6-47
OPEN statement, 6-121
file locking, 5-68
READ statement, 6-152
automatic record locking, 5-71
manual record locking, 5-72
Logic error condition
indexed file, 5-64
relative file, 5-54
sequential file, 5-47
Logical operator
AND, 5-41
glossary term, C-17
meanings, 5-40
NOT, 5-41
OR, 5-41
Logical page, 4-10
glossary term, C-17
Logical record
ACCESS MODE clause, 3-24
BLOCK CONTAINS clause, 4-7
EXTEND phrase, OPEN statement, 6-126
FILLER clause, 4-29
glossary term, C-17

level-number, 4-31
MERGE statement, 6-109
organization input-output, 5-46, 5-53, 5-62
PADDING CHARACTER clause, 3-30
READ statement, 6-146
RECORD clause, 4-16
RELEASE statement, 6-159
RENAMES clause, 4-47
RETURN statement, 6-161
REWRITE statement, 6-163
SAME RECORD AREA clause, 3-39
SORT statement, 6-186
START statement (relative and indexed I-O), 6-193
WRITE statement, 6-212
Low order end, 5-26
glossary term, C-17
LOW phrase
ACCEPT statement, 6-17
DISPLAY statement, 6-64
Lowercase letters, character set, 1-1
LOWLIGHT clause, screen description entry, 4-90
LOWLIGHT phrase, ACCEPT statement, 6-17
LOW-VALUE (LOW-VALUES) figurative constant, 1-16, 3-11
Low-volume-I-O-name
ACCEPT statement, 6-2
DISPLAY statement, 6-58
glossary term, C-17
nonreserved system-names, A-9
SPECIAL-NAMES paragraph, 3-9
system-names, 1-10

M

MAGNETIC-TAPE device-name
ASSIGN clause, 3-25
nonreserved system-names, A-10
system-names, 1-9
Manual multiple
glossary term, C-18
record locking, 3-29, 5-72
MANUAL phrase, LOCK MODE clause, file control entry, 3-28
Manual record locking modes, 3-28, 5-72
glossary term, C-18
Manual single
glossary term, C-18
record locking, 3-28, 5-72
Mass storage control system, glossary term, C-18
Mass storage file, glossary term, C-18
Mass storage, glossary term, C-18
MCS. *See* Message Control System
Memory allocation, 4-57
MEMORY clause, 3-6
Merge
alphabet-name, 3-12
MERGE statement, 6-106
RETURN statement, 6-161
sort-merge file control entry, 3-35
sort-merge file description entry, 4-6
MERGE device-name, system-names, 1-9
Merge file
glossary term, C-18
sort-merge file control entry, 3-35
MERGE statement, 6-106
restrictions, 5-12
Message Control System, 1-5, 4-2, 4-64
communication facility, 5-75
DISABLE statement, 6-55
ENABLE statement, 6-75
glossary term, C-18
interface area, 4-64
PURGE statement, 6-145
RECEIVE statement, 6-155
SEND statement, 6-173
MESSAGE COUNT clause, input CD entry, 4-68
MESSAGE COUNT phrase, ACCEPT statement, 6-24
Message count, glossary term, C-18

MESSAGE DATE clause
input CD entry, 4-68
input-output CD entry, 4-73
Message indicators, 5-79
glossary term, C-18
SEND statement, 6-173
MESSAGE phrase, RECEIVE statement, 6-157
Message segments
communication, 5-79
glossary term, C-19
MESSAGE TIME clause
input CD entry, 4-68
input-output CD entry, 4-73
Messages
communication, 5-79
compiler, B-1
glossary term, C-18
Mnemonic-name
ACCEPT statement, 6-2
DISPLAY statement, 6-58
scope, 1-31
SEND statement, 6-176
SET statement, 6-179
SPECIAL-NAMES paragraph, 3-9
user-defined word type, 1-7
WRITE statement, 6-216
MODE
ACCESS clause, file control entry, 3-23
DEBUGGING clause, SOURCE-COMPUTER paragraph, 3-5
LOCK clause, file control entry, 3-28
Modes
access, 3-23, 5-44, 5-51, 5-60. *See also* Dynamic access, Random access, and Sequential access
file locking, 3-28, 5-68
of operation, arithmetic statements, 5-16
record locking, 3-28, 5-71
MOVE statement, 6-113
MULTIPLE FILE TAPE clause, I-O-CONTROL entry, 3-40
MULTIPLE phrase, LOCK MODE clause, file control entry, 3-29
Multiple record locking modes, 3-29, 5-70, 5-73
glossary term, C-19
MULTIPLY statement, 5-16, 6-119

N

Names
global, 1-27
local, 1-27
Native character set, 3-6
glossary term, C-19
NATIVE code-name, 3-10
Native collating sequence, 3-6
glossary term, C-19
Negated condition, 5-41
negated combined, glossary term, C-19
negated simple, glossary term, C-19
NEGATIVE sign condition, conditional expressions, 5-39
Nested source programs, 1-26
Next executable sentence, 5-7
glossary term, C-19
Next executable statement, 5-7
glossary term, C-19
NEXT operator, constant-expressions, 4-63
NEXT PAGE phrase, WRITE statement, 6-216
NEXT phrase, READ statement, 6-146
Next record
file position indicator, organization input-output, 5-44, 5-51, 5-60
glossary term, C-19
MERGE statement, 6-109
READ statement, 6-147
RETURN statement, 6-161
START statement, 6-194
NEXT SENTENCE phrase
ACCEPT statement, 6-14
IF statement, 6-90
SEARCH statement (ALL), 6-169
SEARCH statement (serial), 6-169

NO ADVANCING phrase, DISPLAY statement, 6-59
NO BEEP phrase, ACCEPT statement, 6-9
NO BELL phrase, ACCEPT statement, 6-9
NO DATA phrase, RECEIVE statement, 6-156
NO HIGHLIGHT clause, screen description entry, 4-90
NO LOCK phrase, READ statement, 6-152
NO REWIND phrase
CLOSE statement, 6-46
OPEN statement, 6-121, 6-126
Noncontiguous items
glossary term, C-19
record description entry, 4-19, 4-24
Nonnumeric item, 6-26
glossary term, C-19
Nonnumeric literals, 1-14
glossary term, C-19
NOT AT END phrase
indexed file, 5-68
READ statement, 6-150
relative file, 5-58
RETURN statement, 6-162
sequential file, 5-50
NOT END-OF-PAGE phrase, WRITE statement, 6-217
NOT ESCAPE phrase
ACCEPT Screen-Name statement, 6-27
ACCEPT statement (terminal I-O), 6-14
NOT EXCEPTION phrase
ACCEPT statement, 6-14
CALL statement, 6-39
NOT INVALID KEY phrase
DELETE statement, 6-52
indexed file, 5-66
READ statement, 6-153
relative file, 5-57
REWRITE statement, 6-165
START statement (relative and indexed I-O), 6-195
WRITE statement, 6-218
NOT logical operator
abbreviated combined relation conditions, 5-42
constant-expressions, 4-63
negated conditions, 5-41
NOT OPTIONAL phrase, SELECT clause, file control entry, 3-23
NOT OVERFLOW phrase
STRING statement, 6-201
UNSTRING statement, 6-210
NOT SIZE ERROR phrase
ADD statement, 6-30
common rules, 5-18
COMPUTE statement, 6-48
DIVIDE statement, 6-73
MULTIPLY statement, 6-120
SUBTRACT statement, 6-204
NULL (NULLS) figurative constant, 1-16, 5-25, 6-36, 6-178
Null, glossary term, C-19
Numeric character, 4-35, 4-38, 4-39
glossary term, C-20
PICTURE character-string, 4-42
user-defined words, 1-4
zero suppression editing, 4-43
Numeric class, 4-95
NUMERIC class condition
COMPUTATIONAL usage, 4-53
COMPUTATIONAL-3 usage, 4-54
conditional expressions, 5-37
DISPLAY usage, 4-56
Numeric data item, 4-35
Numeric edited data item, 4-36
Numeric item
glossary term, C-20
operational sign, 4-48
Numeric literals, 1-14
glossary term, C-20
NUMERIC phrase, INITIALIZE statement, 6-92
NUMERIC-EDITED phrase, INITIALIZE statement, 6-92

O

Object computer entry, 3-6
  glossary term, C-20
Object of entry, glossary term, C-20
Object program, 2-1, 3-1, 4-1, 5-1, 5-76
  glossary term, C-20
Object time, glossary term, C-20
OBJECT-COMPUTER paragraph, 3-6
  glossary term, C-20
Objects
  external, 1-28
  internal, 1-28
OBJECT-VERSION keyword, COMPILER-OPTIONS configuration record, B-84
Obsolete element, glossary term, C-20
OCCURS clause, data description entry, 1-11, 4-24, 4-32, 4-63
OFF phrase
  ACCEPT statement, 6-17
  REPLACE statement, 1-40
  SET statement, 6-178
OMITTED phrase
  CALL PROGRAM statement, 6-42
  CALL statement, 6-36
  LABEL RECORDS clause, 4-9
ON phrase, SET statement, 6-178
Open mode
  glossary term, C-20
  permissible statements, 6-123
OPEN statement, 6-121
Operand
  glossary term, C-20
  overlapping operands, 5-19
Operational sign, 1-13
  glossary term, C-20
Optional file
  glossary term, C-21
  I-O status, 5-46, 5-53, 5-62, 6-124
OPTIONAL phrase, SELECT clause, file control entry, 3-22
Optional words, 1-10
  glossary term, C-21
OR logical operator
  abbreviated combined relation conditions, 5-42
  combined conditions, 5-41
  constant-expressions, 4-63
Organization
  indexed file, 5-59
  relative file, 5-50
  sequential file, 5-44
ORGANIZATION clause, file control entry, 3-29
Organization of this guide, xix
Output code set, 3-12
OUTPUT device-name
  ASSIGN clause, 3-25
  system-names, 1-9
Output file, 3-9
  glossary term, C-21
  MERGE statement, 6-106
  SORT statement, 6-183
Output mode
  glossary term, C-21
  OPEN statement, 6-125
OUTPUT phrase
  DISABLE statement, 6-56
  ENABLE statement, 6-76
  OPEN statement, 6-125
  USE statement, 5-14
Output procedure
  glossary term, C-21
  MERGE statement, 5-8, 5-10, 5-12, 5-75, 6-106
  SORT statement, 5-8, 5-10, 5-12, 5-75, 6-183
OUTPUT PROCEDURE phrase
  MERGE statement, 6-109
  SORT statement, 6-187
OVERFLOW phrase
  CALL statement, 6-39
  STRING statement, 6-201
  UNSTRING statement, 6-210
Overlapping operands, 5-19

P

PACKED-DECIMAL usage, data
description entry, 4-54
PADDING CHARACTER clause, file
control entry, 3-30
Padding characters, 3-30
glossary term, C-21
Page body, 4-11
glossary term, C-21
PAGE phrase
SEND statement, 6-177
WRITE statement, 6-216
Paragraph, 1-24, 5-6
glossary term, C-21
Paragraph header
AUTHOR, 2-1
DATE-COMPILED, 2-1
DATE-WRITTEN, 2-1
FILE-CONTROL, 3-4
glossary term, C-21
INSTALLATION, 2-1
I-O-CONTROL, 3-4
OBJECT-COMPUTER, 3-2
PROGRAM-ID, 2-1
REMARKS, 2-1
SECURITY, 2-1
SOURCE-COMPUTER, 3-2
SPECIAL-NAMES, 3-2
PARAGRAPH phrase, EXIT
PARAGRAPH statement, 6-85
Paragraph-name
glossary term, C-22
Procedure Division paragraph, 5-4
qualification, 5-7
scope, 1-30
user-defined word type, 1-7
Parentheses
arithmetic expressions, 1-2, 5-20, 5-22
as separators, 1-2
binary allocation override, 1-2, 4-55
conditions, 1-2, 5-23
constant-expressions, 1-2, 4-62
reference modifiers, 1-2, 4-102
subscripts, 1-2, 4-100
Patterns
glossary term, C-22
LIKE relation condition, 5-25
regular expressions, 5-29
PERFORM phrase, EXIT PERFORM
statement, 6-85
PERFORM statement, 6-128
conditional expressions, 5-23
restrictions, 5-12
Period, 3-18
DECIMAL POINT IS COMMA clause,
SPECIAL-NAMES paragraph, 3-18
numeric literals, 1-14
PICTURE character-string, 4-39
Permanent error condition
indexed file, 5-63
relative file, 5-54
sequential file, 5-46
Permanent segments, 3-7
Phrase
conditional, 1-22
glossary term, C-22
Physical page, 4-10, 6-216
glossary term, C-22
Physical record, 4-7, 5-69
glossary term, C-22
PICTURE character-strings, 1-17
in PICTURE clause (data description
entry), 4-34
in PICTURE clause (screen description
entry), 4-92
PICTURE clause
data description entry, 4-26, 4-34
editing rules, 4-40
screen description entry, 4-92

Picture symbol precedence, 4-44
Pointer data items
  comparison, 5-27
  data description entry, 4-57
  glossary term, C-22
  INITIALIZE statement, 6-92
  LIKE relation condition, 5-28
  NULL (NULLS) figurative constant, 1-16
  pattern, 5-28
  SET statement, 6-178
  usage, 4-57
POINTER phrase
  STRING statement, 6-201
  UNSTRING statement, 6-208
POINTER usage, data description entry, 4-57
POSITION phrase
  ACCEPT statement (terminal I-O), 6-17
  DISPLAY statement (terminal I-O), 6-65
  MULTIPLE FILE TAPE clause, I-O-CONTROL entry, 3-40
POSITIVE sign condition, conditional expressions, 5-39
PREVIOUS phrase, READ statement, 6-146
Previous record, glossary term, C-22
Prime record key
  DUPLICATES phrase, REWRITE statement, 6-164
  glossary term, C-22
  KEY phrase, READ statement, 6-151
  OPEN statement
    INPUT phrase, 6-124
    I-O phrase, 6-125
  ORGANIZATION clause, file control enry, 3-30
  RECORD KEY clause, file control enry, 3-32
  RECORD KEY clause, file control entry, 5-59
  START statement (relative and indexed I-O), 6-192
    SIZE phrase, 6-195
  WRITE statement, 6-214
PRINT device-name
  ASSIGN clause, 3-25
  nonreserved system-names, A-10
  system-names, 1-9
PRINTER device-name
  ASSIGN clause, 3-25
  nonreserved system-names, A-10
  system-names, 1-9
PRINTER-1 device-name
  ASSIGN clause, 3-25
  nonreserved system-names, A-10
  system-names, 1-9
Procedure branching statements
  ALTER, 6-32
  CALL, 6-33
  CALL PROGRAM, 6-41
  EXIT, 6-84
  EXIT PROGRAM, 6-84
  glossary term, C-22
  GO TO, 6-88
  GOBACK, 6-87
  MERGE, 6-106
  PERFORM, 6-128
  SORT, 6-183
Procedure Division
  common rules, 5-16
  communication facility, 5-75
    message segments, 5-79
    messages, 5-79
  conditional expressions, 5-23
  header
    GIVING phrase, 5-3
    RETURNING phrase, 5-2
    USING phrase, 5-1
  indexed input/output, 5-59
  interactive input/output, 5-74
  order of execution, 5-6
  paragraph, 5-6
  procedure references, 5-7
  procedures, 5-6
  record locking, 5-69
  relative input/output, 5-50
  section, 5-6

segmentation, 5-9
sequential input/output, 5-43
sort-merge, 5-75
statement, 5-6
structure, 5-4
transfers of control, 5-7
explicit, 5-8
implicit, 5-7
Procedure references, 5-7
Procedure-name
ALTER statement, 6-32
glossary term, C-22
GO TO statement, 6-88
MERGE statement, 6-106
PERFORM statement, 6-128
procedures, 5-6
SORT statement, 6-183
Procedures, 5-6
glossary term, C-22
Program
common, 1-29
initial, 1-29
Program collating sequence, 3-6
MERGE statement, 6-108
SORT statement, 6-185
PROGRAM COLLATING SEQUENCE
clause, 3-6
Program identification entry, 2-2
glossary term, C-22
PROGRAM phrase, EXIT PROGRAM
statement, 6-84
Program structure, 1-18
sentences, imperative, 1-24
statements, verbs, 1-21
PROGRAM-ID paragraph, 2-2
PROGRAM-ID special register, 1-12
Program-name, 1-31
CALL PROGRAM statement, 6-41
CALL statement, 6-34
CANCEL statement, 6-43
END PROGRAM header, 1-34
glossary term, C-23
PROGRAM-ID paragraph, 1-12, 2-2
scope, 1-31
user-defined word type, 1-7
PROMPT phrase, ACCEPT statement, 6-19
Pseudo-text
COPY statement, 1-35, 1-36
glossary term, C-23
REPLACE statement, 1-40, 1-41
Pseudo-text delimiter, 1-3
COPY statement, 1-35
glossary term, C-23
REPLACE statement, 1-40
Punctuation
characters, 1-2, 1-17, 5-31, 5-32
Punctuation character, glossary term, C-23
PURGE statement, 6-145

Q

Qualification, 1-32, 4-97
condition-names, 4-98
data-names, 4-98
LINAGE-COUNTER, 4-98
paragraph-names, 5-7
screen-name, 4-98
split-key-name, 4-98
text-name, 1-35–1-36
Qualified data-name, 4-99
glossary term, C-23
Qualifier, glossary term, C-23
QUEUE clause, input CD entry, 4-67
Queue hierarchy, 5-81
Queue name, 5-81
glossary term, C-24
Queues
communication, 5-80
glossary term, C-24

Quotation marks, using as separators, 1-3
QUOTE (QUOTES) figurative
  constant, 1-16

R

Random access
  glossary term, C-24
  mode
    for indexed file organization, 5-60
    for relative file organization, 5-51
RANDOM device-name
  ASSIGN clause, 3-25
  system-names, 1-9
READ statement, 4-17, 6-146
RECEIVE statement, 6-155
Record area, 4-15
  glossary term, C-24
RECORD clause, file description entry, 4-15
RECORD DELIMITER clause, file control
  entry, 3-31
Record delimiting techniques, 1-8
  binary sequential, 3-31
  glossary term, C-24
  line sequential, 3-31
  STANDARD-1, 3-31
Record description entry, 4-23
  Communication Section, 4-22
  File Section, 4-4
  glossary term, C-24
  Linkage Section, 4-19
  Working-Storage Section, 4-18
Record description, glossary term, C-24
RECORD KEY clause, file control
  entry, 3-32
Record key, glossary term, C-24
Record locking, 5-69
  CLOSE statement, 6-47
  DELETE statement, 6-52
  LOCK MODE clause, file control
    entry, 3-28
  modes
    automatic, 3-28, 5-71
    manual, 3-28, 5-72
    multiple, 3-29, 5-73
    single, 3-28, 5-73
  READ statement, 6-152
  REWRITE statement, 6-164
  UNLOCK statement, 6-206
  WRITE statement, 6-213
Record locking mode
  automatic, 3-28, 5-71
  glossary term, C-24
  manual, 3-28, 5-72
  multiple, 3-29, 5-73
  single, 3-28, 5-73
Record number, glossary term, C-25
Record, glossary term, C-24
Record-name
  global, 1-27
  glossary term, C-25
  RELEASE statement (sort), 6-159
  REWRITE statement, 6-163
  scope, 1-32
  user-defined word type, 1-7
  WRITE statement, 6-213
REDEFINES clause, data description
  entry, 4-46
REEL phrase, CLOSE statement, 6-46
Reel, glossary term, C-25
Reference modification, 4-102, 5-19
Reference modifiers
  glossary term, C-25
  parentheses, using in, 1-2, 4-102
References to table items, 4-106
Regular expressions, 5-29
  glossary term, C-25
Related publications, xxi
Relation character, glossary term, C-25
Relation condition, 4-61, 5-24
  glossary term, C-25
Relation, glossary term, C-25

Relational operator, 1-13, 4-61
condition-name, 4-26
glossary term, C-26
meanings, 5-25
SEARCH statement (ALL), 6-167
START (relative and indexed I-O) statement, 6-191
Relationship of the object program to the message control system and communication devices, 5-76
Relative file
access modes, 5-51
file control entry, 3-20
glossary term, C-26
Relative key
file control entry, 3-23
glossary term, C-26
RELATIVE KEY phrase, ACCESS MODE clause, file control entry, 3-23
Relative organization input-output, 5-50
CLOSE statement, 6-45
DELETE FILE statement, 6-54
DELETE statement, 6-51
file description entry, 4-5
OPEN statement, 6-121
READ statement, 6-146
relative file control entry, 3-20
REWRITE statement, 6-163
START statement, 6-191
UNLOCK statement, 6-206
WRITE statement, 6-212
Relative organization, glossary term, C-26
Relative record number, glossary term, C-26
RELEASE statement, 4-16, 4-34, 6-159
REMAINDER phrase, DIVIDE statement, 6-73
REMARKS paragraph, 2-3
REMOVAL phrase, CLOSE statement, 6-47
RENAMES clause, data description entry, 4-26, 4-47
REPLACE statement, 1-40
REPLACING phrase
COPY statement, 1-35, 1-37
INITIALIZE statement, 6-92
INSPECT statement, 6-97
SEND statement, 6-177
REQUIRED clause, screen description entry, 4-93
RERUN clause, I-O-CONTROL entry, 3-36
Rerun-name, RERUN clause, 3-4, 3-19, 3-36
RESERVE clause, file control entry, 3-34
Reserved words, 1-4, 1-10
context-sensitive, 1-14, A-7
glossary term, C-26
list of, A-1–A-6
special symbols, A-8
Resource, glossary term, C-26
Restrictions on program flow, 5-11
Resultant identifier, glossary term, C-26
RETURN statement, 4-17, 6-161
RETURN-CODE special register, 1-13
RETURNING phrase
CALL statement, 6-38
Procedure Division header, 5-2
REVERSE clause, screen description entry, 4-93
REVERSE phrase
ACCEPT statement, 6-19
DISPLAY statement, 6-66
REVERSED clause, screen description entry, 4-93
REVERSED phrase
ACCEPT statement, 6-19
DISPLAY statement, 6-66
OPEN statement, 6-121, 6-124
REVERSE-VIDEO clause
DISPLAY statement, 6-66
screen description entry, 4-93
REVERSE-VIDEO phrase, ACCEPT statement, 6-19
REWIND phrase. *See* NO REWIND phrase
REWRITE statement, 4-16, 4-34, 6-163

ROUNDED phrase, 5-17
ADD statement, 6-30
common rules, 5-17
COMPUTE statement, 6-48
DIVIDE statement, 6-73
MULTIPLY statement, 6-119
SUBTRACT statement, 6-204
Routine-name
ENTER statement, 6-78
glossary term, C-26
user-defined word type, 1-7
RUN phrase, STOP statement, 6-197
Run unit
CALL PROGRAM statement, 6-41
CALL statement, 6-33
glossary term, C-26
inter-program communication, 1-26
RUN-FILES-ATTR configuration record,
FORCE-USER-MODE keyword,
5-68–5-70

S

SAME clause, I-O-CONTROL entry, 3-38
Scheduled initiation of the object
program, 5-78
Scope of
names, 1-30
statements, 1-23
Scope terminator, 1-22–1-23
Screen clause
AUTO, 4-84
BACKGROUND, 4-84
BACKGROUND-COLOR, 4-84
BELL, 4-85
BLANK LINE, 4-85
BLANK REMAINDER, 4-86
BLANK SCREEN, 4-86
BLANK WHEN ZERO, 4-86
BLINK, 4-87
COLUMN, 4-87
ERASE, 4-88
FOREGROUND, 4-89
FOREGROUND-COLOR, 4-89
FULL, 4-89
glossary term, C-27
HIGHLIGHT, 4-90
JUSTIFIED, 4-90
LINE, 4-90
LOWLIGHT, 4-90
PICTURE, 4-92
REQUIRED, 4-93
REVERSE, 4-93
SECURE, 4-93
SIGN, 4-94
UNDERLINE, 4-94
USAGE, 4-94
VALUE, 4-95
Screen description entry, 4-80
AUTO clause, 4-84
BACKGROUND clause, 4-84
BACKGROUND-COLOR clause, 4-84
BELL clause, 4-85
BLANK LINE clause, 4-85
BLANK REMAINDER clause, 4-86
BLANK SCREEN clause, 4-86
BLANK WHEN ZERO clause, 4-86
BLINK clause, 4-87
COLUMN clause, 4-87
ERASE clause, 4-88
FOREGROUND clause, 4-89
FOREGROUND-COLOR clause, 4-89
FULL clause, 4-89
glossary term, C-27
HIGHLIGHT clause, 4-90
JUSTIFIED clause, 4-90
LINE clause, 4-90
LOWLIGHT, 4-90
PICTURE clause, 4-92
REQUIRED clause, 4-93
REVERSE clause, 4-93
SECURE clause, 4-93
SIGN clause, 4-94
UNDERLINE clause, 4-94
USAGE clause, 4-94
VALUE clause, 4-95
Screen item, glossary term, C-27

Screen Section
Data Division, 4-2, 4-22
glossary term, C-27
Screen-name
ACCEPT Screen-Name statement, 6-25
DISPLAY Screen-Name statement, 6-69
glossary term, C-27
qualification, 4-97
scope, 1-30
user-defined word type, 1-7
SD. *See* Sort-merge file description entry
SEARCH statement, 5-23, 6-167
Section header
Data Division
Communication Section, 4-3, 4-22
File Section, 4-3, 4-4
Linkage Section, 4-3, 4-19
Screen Section, 4-3, 4-22
Working-Storage Section, 4-3, 4-18
Environment Division
Configuration Section, 3-2, 3-5
Input-Output Section, 3-4, 3-18
glossary term, C-27
Procedure Division, section-name
section, 5-4
SECTION phrase
Communication Section header, 4-22
Configuration Section header, 3-5
EXIT SECTION statement, 6-85
File Section header, 4-4
Input-Output Section header, 3-18
Linkage Section header, 4-19
Procedure Division Section header, 5-1
Screen Section header, 4-22
Working-Storage Section header, 4-18
Section-name
glossary term, C-27
qualification, 5-7
scope, 1-30
section header, 5-4
user-defined word type, 1-7
Sections, 1-24
glossary term, C-27
SECURE clause, screen description
entry, 4-93
SECURE phrase, ACCEPT statement, 6-17
SECURITY paragraph, 2-3
SEGMENT phrase, RECEIVE
statement, 6-157
Segmentation, 5-9
Segmentation classification, 5-11
SEGMENT-LIMIT clause, 3-7
Segment-number
glossary term, C-28
section header, 5-4
segmentation, 5-9
SEGMENT-LIMIT clause, 3-7
user-defined word type, 1-8
Segments, 5-9
message, 5-79
SELECT clause
file control entry, 3-21
sort-merge file control entry, 3-35
Selection object, EVALUATE
statement, 6-80
Selection subject, EVALUATE
statement, 6-80
SEND statement, 6-173
Sentences, 1-24
conditional, 1-24
directive, 1-24
glossary term, C-28
imperative, 1-24
SEPARATE CHARACTER phrase, SIGN
clause, 4-48
Separately compiled program, glossary
term, C-28
Separators
colons, 1-3
glossary term, C-28
list of, 1-1
parentheses, 1-2
pseudo-text delimiter, 1-3
quotation marks, 1-3
rules for forming, 1-2

SEQUENCE clause
indexed file COLLATING, 3-27
MERGE COLLATING, 6-108
PROGRAM COLLATING, 3-6
SORT COLLATING, 6-185
Sequence number, illustrated, 1-18
Sequential access
glossary term, C-28
mode
for indexed file organization, 5-60
for relative file organization, 5-51
for sequential file organization, 5-44
Sequential file
access modes, 5-44
file control entry, 3-20
glossary term, C-28
Sequential organization input-output, 5-43
CLOSE statement, 6-45
DELETE FILE statement, 6-54
file description entry, 4-5
OPEN statement, 6-121
READ statement, 6-146
REWRITE statement, 6-163
sequential file control entry, 3-20
UNLOCK statement, 6-206
WRITE statement, 6-212
Sequential organization, glossary term, C-28
SET statement, 6-178
Shared file, 5-69
glossary term, C-28
Shared file environment, 5-69–5-70
glossary term, C-28
Shared mode
file locking, 3-28, 5-68, 5-69
glossary term, C-28
Sharing in a run unit
data, 1-29
files, 1-30
SIGN clause
data description entry, 4-48
screen description entry, 4-94
Sign condition, 5-39
glossary term, C-28
Simple condition, 5-23
glossary term, C-29
Simple insertion editing, 4-41
Single record locking modes, 3-28, 5-70, 5-73
glossary term, C-29
Size error condition
ADD statement, 6-30
common rules, 5-18
COMPUTER statement, 6-48
DIVIDE statement, 6-73
MULTIPLY statement, 6-120
SUBTRACT statement, 6-204
SIZE ERROR phrase
ADD statement, 6-30
common rules, 5-18
COMPUTE statement, 6-48
DIVIDE statement, 6-73
MULTIPLY statement, 6-120
SUBTRACT statement, 6-204
SIZE operator, constant-expressions, 4-63
SIZE phrase
ACCEPT statement, 6-19
DISPLAY statement, 6-67
START statement (relative and indexed I-O), 6-195
SIZE, MEMORY clause, 3-6
SORT device-name, system-names, 1-9
Sort file, 3-35
glossary term, C-29
SORT statement, 6-183
restrictions, 5-12
Sort-merge, 5-75
alphabet-name, 3-12
MERGE statement, 6-106
RELEASE statement, 6-159
RETURN statement, 6-161
SORT statement, 6-183
sort-merge file control entry, 3-35
sort-merge file description entry, 4-6
SORT-MERGE device-name, system-names, 1-9

Sort-merge file control entry
ASSIGN clause, 3-35
SELECT clause, 3-35
Sort-merge file description entry, 4-6
DATA RECORDS clause, 4-8
glossary term, C-29
RECORD clause, 4-15
SORT-WORK device-name
ASSIGN clause, 3-25
nonreserved system-names, A-10
system-names, 1-9
SOURCE clause, input CD entry, 4-68
Source computer entry, glossary term, C-29
Source format, 1-18
glossary term, C-29
Source program
general format, 1-25
glossary term, C-29
Source, glossary term, C-29
SOURCE-COMPUTER paragraph, 3-5
glossary term, C-29
SPACE (SPACES) figurative constant, 1-16
Special character word, glossary term, C-30
Special characters, 1-13
glossary term, C-30
Special insertion editing, 4-41
Special names entry, glossary term, C-30
Special registers, 1-11, 4-97
ADDRESS, 1-11, 4-19, 5-4, 5-25, 6-36, 6-38
COUNT, 1-11
COUNT-MAX, 1-11
COUNT-MIN, 1-12
glossary term, C-30
LENGTH, 1-12, 6-182
LINAGE-COUNTER, 1-12, 4-6, 4-12, 4-99
PROGRAM-ID, 1-12
RETURN-CODE, 1-13
Special symbols, A-8
SPECIAL-NAMES paragraph, 3-7
glossary term, C-30
Split key, glossary term, C-30
Split-key-name
ALTERNATE RECORD KEY clause, 3-32
global, 1-27, 4-9
glossary term, C-31
READ statement, 6-146, 6-151
RECORD KEY clause, 3-32
scope, 1-32
START statement, 6-191
user-defined word type, 1-8
Standard alignment rules, 4-96
Standard data format, glossary term, C-31
STANDARD phrase, LABEL RECORDS clause, 4-9
STANDARD-1
code-name, 3-10
record delimiting technique, 3-31
STANDARD-2, code-name, 3-10
START operator, constant-expressions, 4-63
START statement (relative and indexed I-O), 6-191
Statements, 1-21
conditional, 1-21
delimited scope, 1-21–1-24
directive, 1-21, 1-24
glossary term, C-31
imperative, 1-21–1-23
nesting, 1-23
STATUS clause. *See* FILE STATUS clause
STATUS KEY clause
input CD entry, 4-68
input-output CD entry, 4-74
output CD entry, 4-72
Status values. *See* I-O status values
STOP statement, 6-197
STRING statement, 6-199
Subject of entry, glossary term, C-31
Subprogram
CALL statement, 6-34
glossary term, C-31
Sub-queue, glossary term, C-31
SUB-QUEUE-1 clause, input CD entry, 4-67

SUB-QUEUE-2 clause, input CD
entry, 4-67
SUB-QUEUE-3 clause, input CD
entry, 4-68
Subscripted data item, glossary term, C-31
Subscripting, 4-100
Subscripts, 4-100
evaluation, 5-16
glossary term, C-31
parentheses, using in, 1-2, 4-100
SUBTRACT statement, 5-16, 6-203
SUPPRESS-FILLER-IN-SYMBOL-TABLE
keyword, COMPILER-OPTIONS
configuration record, B-93
Suppression editing, 4-43
SUPPRESS-LITERAL-BY-CONTENT
keyword, COMPILER-OPTIONS
configuration record, 6-37
SWITCH-1–SWITCH-8 switch-names
nonreserved system-names, A-3
SPECIAL-NAMES paragraph, 3-9
system-names, 1-10
Switch-name, 1-8
glossary term, C-31
list of, 1-9
SPECIAL-NAMES paragraph, 3-9
SWITCH-1–SWITCH-8 and
UPSI-0–UPSI-7, A-9
Switch-status condition, 5-40
glossary term, C-31
Switch-status, SET statement
(ON/OFF), 6-179, 6-181
SYMBOLIC CHARACTERS clause, 3-15
SYMBOLIC DESTINATION clause,
output CD entry, 4-72
SYMBOLIC QUEUE clause, input CD
entry, 4-67
SYMBOLIC SOURCE clause, input CD
entry, 4-68
SYMBOLIC SUB-QUEUE-1 clause,
input CD entry, 4-67
SYMBOLIC SUB-QUEUE-2 clause,
input CD entry, 4-67
SYMBOLIC SUB-QUEUE-3 clause,
input CD entry, 4-68
SYMBOLIC TERMINAL clause,
input-output CD entry, 4-74
Symbolic-character
figurative constant, 1-16
glossary term, C-31
scope, 1-31
SYMBOLIC CHARACTERS clause,
3-15
user-defined word type, 1-8
Symbols and conventions, xviii
Symbols, special, A-8
SYMBOL-TABLE-OUTPUT keyword,
COMPILER-OPTIONS configuration
record, B-84
SYNCHRONIZED clause, data description
entry, 4-50
SYSIN low-volume-I-O-name
ACCEPT statement, 6-2
nonreserved system-names, A-9
SPECIAL-NAMES, 3-9
system-names, 1-10
SYSOUT low-volume-I-O-name
DISPLAY statement, 6-58
nonreserved system-names, A-9
SPECIAL-NAMES, 3-9
system-names, 1-10
System-names
defined, 1-4, 1-8
file control entry, A-10
glossary term, C-31
list of, 1-9
nonreserved, list of, A-9–A-11
SPECIAL-NAMES paragraph, A-9

T

TAB phrase, ACCEPT statement, 6-20
Table definition, 4-104
Table element, glossary term, C-32
Table handling, 4-104
Table items, referencing, 4-106
Table, glossary term, C-31

Table-name, glossary term, C-32
TALLYING phrase
  INSPECT statement, 6-97
  UNSTRING statement, 6-209
TAPE device-name
  ASSIGN clause, 3-25
  system-names, 1-9
TERMINAL clause, input-output CD entry, 4-74
TERMINAL phrase
  DISABLE statement, 6-57
  ENABLE statement, 6-77
Terminal, glossary term, C-32
TEST AFTER phrase
  PERFORM statement, 6-129
  PERFORM UNTIL statement, 6-132
  PERFORM VARYING AFTER statement, 6-139
  PERFORM VARYING statement, 6-137
TEST BEFORE phrase
  PERFORM statement, 6-129
  PERFORM UNTIL statement, 6-132
  PERFORM VARYING AFTER statement, 6-134
  PERFORM VARYING statement, 6-133
TEXT LENGTH clause
  input CD entry, 4-68
  input-output CD entry, 4-74
  output CD entry, 4-72
Text word, 1-36
  glossary term, C-32
Text-name
  COPY statement, 1-35
  glossary term, C-32
  qualification, 1-35
  scope, 1-30
  user-defined word type, 1-8
THROUGH phrase
  EVALUATE statement, 6-80
  PERFORM statement, 6-131
Time of day
  ACCEPT FROM TIME statement, 6-5
  DATE-AND-TIME phrase, ACCEPT FROM statement, 6-4
  DAY-AND-TIME phrase, ACCEPT FROM statement, 6-4
TIME phrase, ACCEPT statement, 6-21
Time-out, ACCEPT statement, TIME phrase, 6-21
TIMES phrase, PERFORM statement, 6-131
TO LINE phrase, WRITE statement, 6-216
TO phrase, PICTURE clause, screen description entry, 4-92
TO VALUE phrase, INITIALIZE statement, 6-93
Top margin, 4-11
  glossary term, C-32
TRAILING phrase, SIGN clause, 4-48
Transfers of control, explicit and implicit, 5-7
Translation
  CODE-SET clause (sequential I-O), 3-26
  EBCDIC, 3-15
TRIMMED phrase, LIKE relation condition, 5-28
TRUE phrase, SET statement, 6-178
TRUE, selection subject or object, 6-81
Truth value, 5-23
  glossary term, C-32

U

Unary operator, 5-20–5-23
  glossary term, C-32
UNDERLINE clause, screen description entry, 4-94
Unicode
  glossary term, C-32
  LIKE condition, 5-29, 5-32–5-33
Uniqueness of reference
  qualification, 1-32, 4-97
  reference-modification, 4-102
  subscripting, 4-100

UNIT phrase
ACCEPT statement, 6-21
CLOSE statement, 6-46
DISPLAY statement, 6-67
Unit, glossary term, C-32
UNLOCK statement, 6-206
UNSTRING statement, 6-207
Unsuccessful execution, glossary term, C-32
UNTIL phrase, PERFORM statement, 6-132
UP BY phrase, SET statement, 6-178
UPDATE phrase, ACCEPT statement, 6-22
Uppercase letters, character set, 1-1
UPSI-0–UPSI-7 switch-names
nonreserved system-names, A-9
SPECIAL-NAMES paragraph, 3-9
system-names, 1-10
USAGE clause
data description entry, 4-52, 5-2
screen description entry, 4-94
USE statement, 5-13
User-defined words
glossary term, C-33
rules, 1-5
types, 1-4
USING phrase
CALL PROGRAM statement, 6-42
CALL statement, 6-36
MERGE statement, 6-108
PICTURE clause, screen description entry, 4-92
Procedure Division header, 5-1
SORT statement, 6-186

V

VALUE clause
data description entry, 4-58
screen description entry, 4-95
VALUE OF clause, file description entry, 4-17
VALUE phrase, INITIALIZE statement, 4-58, 6-93
Variable length, 4-33
Variable, glossary term, C-33
Variable-length record, 4-16
glossary term, C-33
Variable-occurrence data item, 4-33
glossary term, C-33
VARYING phrase
PERFORM statement, 6-132
RECORD clause, 4-15
SEARCH statement (serial), 6-170
Verbs, 1-21
conditional, 1-22
glossary term, C-33
imperative, 1-23
Volume, 6-46
glossary term, C-33

W

WHEN OTHER phrase, EVALUATE statement, 6-82
WHEN phrase
EVALUATE statement, 6-80
SEARCH statement (ALL), 6-169
SEARCH statement (serial), 6-168
WITH DATA phrase, RECEIVE statement, 6-156
WITH DEBUGGING MODE clause, SOURCE-COMPUTER paragraph, 3-5
WITH FILLER phrase, INITIALIZE statement, 6-93
WITH KEY phrase
DISABLE statement, 6-57
ENABLE statement, 6-77
WITH LOCK phrase
CLOSE statement, 6-47
OPEN statement, 6-121
READ statement, 6-152
WITH NO ADVANCING phrase, DISPLAY statement, 6-58
WITH NO LOCK phrase, READ statement, 6-152
WITH NO REWIND phrase
CLOSE statement, 6-46
OPEN statement, 6-126

Words
COBOL, defined, 1-4
context-sensitive, 1-4, 1-14, A-7
glossary term, C-33
reserved, 1-4, 1-10, A-1–A-6
system-names, 1-4, 1-8, A-9–A-11
user-defined, 1-4, 1-5–1-8
Working-Storage Section
Data Division, 4-1, 4-18
glossary term, C-33
WRITE statement, 4-16, 4-34, 6-212

X

XML
glossary term, C-33
LIKE relation condition, 5-29–5-32
XML schema
glossary term, C-33
LIKE relation condition, 5-29–5-32

Y

YYYYDDD phrase, ACCEPT FROM
statement, 6-3–6-4
context-sensitive words, A-7
YYYYMMDDD phrase, ACCEPT FROM
statement, 6-3–6-4
context-sensitive words, A-7

Z

ZERO (ZERO, ZEROES) figurative
constant, 1-15
ZERO sign condition, conditional
expressions, 5-39
Zero suppression editing, 4-43
Zero-length item, glossary term, C-33