

SC21-7718-7

File No. S38-24

IBM System/38

IBM System/38 COBOL Reference Manual and Programmer's Guide

Program Number 5714-CB1



SC21-7718-7

File No. S38-24

IBM System/38

IBM System/38 COBOL Reference Manual and Programmer's Guide

Program Number 5714-CB1

Eighth Edition (November 1986)

This major revision obsoletes SC21-7718-6.

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

This edition applies to Release 8, Modification Level 0, of IBM System/38 COBOL Program Product (Program 5714-CB1) and to all subsequent releases and modifications until otherwise indicated in new editions.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to

IBM Canada Ltd.
Information Development,
Department 849,
895 Don Mills Road,
North York, Ontario, Canada. M3C 1W3

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1980, 1981, 1982, 1983, 1984, 1985, 1986

Contents

About This Manual	xi	Debugging Lines	2-16
Purpose of This Manual	xi	Blank Lines	2-16
Organization of This Manual	xi	Overall Punctuation Rules	2-16
Summary of Changes	xii	Identification Division	2-16
What You Should Know	xiii	Environment Division	2-17
If You Need More Information	xiii	Data Division	2-17
Industry Standards	xiv	Procedure Division	2-17
Acknowledgment	xv	Methods of Data Reference	2-17
Chapter 1. Introduction	1-1	Qualification	2-17
General Description	1-1	Qualification Rules	2-19
Language Level	1-1	Subscripting and Indexing	2-20
Compiler Features	1-3	Identifier	2-20
Format Notation	1-4	Condition-Name	2-21
Chapter 2. Language Considerations ..	2-1	Explicit and Implicit References	2-21
COBOL Program Structure	2-1	Data Attribute Specification	2-21
The COBOL Divisions	2-1	Procedure Division Data References	2-22
Clauses and Statements	2-2	Transfers of Control	2-22
Clause and Statement Specification		Chapter 3. Identification and	
Order	2-2	Environment Divisions	3-1
Structure of the Language	2-3	IDENTIFICATION DIVISION	3-1
Character-Strings	2-4	Coding Example	3-2
COBOL Words	2-4	PROGRAM-ID Paragraph	3-2
Literals	2-8	Other Optional Paragraphs	3-3
PICTURE Character-Strings	2-10	ENVIRONMENT DIVISION	3-3
Comments	2-10	Coding Example	3-4
Separators	2-10	Configuration Section	3-4
Standard COBOL Format	2-11	SOURCE-COMPUTER Paragraph	3-6
Sequence Numbers (Columns 1-6) ..	2-11	OBJECT-COMPUTER Paragraph	3-6
Continuation Area (Column 7)	2-12	MEMORY SIZE Clause	3-6
Area A (Columns 8-11) and Area B		PROGRAM COLLATING SEQUENCE	
(Columns 12-72)	2-12	Clause	3-6
Special Considerations	2-14	SEGMENT-LIMIT Clause	3-6
Division Header	2-14	SPECIAL-NAMES Paragraph	3-7
Section Header	2-14	Function-Name-1 Clause	3-7
Paragraph Header, Paragraph-Name	2-14	Function-Name-2 Clause	3-8
Data Division Entries	2-14	Coding Example	3-9
DECLARATIVES and END		Alphabet-Name Clause	3-9
DECLARATIVES	2-15	CURRENCY SIGN Clause	3-12
Program Spacing	2-15	DECIMAL-POINT IS COMMA	
Indentation	2-15	Clause	3-12
Continuation of Lines	2-15	Input-Output Section	3-12
Comment Lines	2-16	Files	3-12
		Data Base Files	3-12

Device Files	3-13	Signed Data	4-22
DDM Files	3-13	Operational Signs	4-22
Paragraphs	3-13	Editing Signs	4-22
File Processing Summary	3-14	Record Description Entry	4-23
Data Organization	3-14	Data Description Entry	4-23
Access Modes	3-15	Format 1	4-25
Access Mode Allowed for Each File		Format 2	4-26
Type	3-16	Format 3	4-26
FILE-CONTROL Paragraph	3-16	Format 4 – Boolean Data	4-27
FILE-CONTROL Paragraph – General		Level-Numbers	4-27
Considerations	3-18	Data-Name or FILLER Clause	4-27
SELECT Clause	3-19	REDEFINES Clause	4-28
ASSIGN Clause	3-19	USAGE Clause	4-31
RESERVE Clause	3-21	DISPLAY Phrase	4-31
ORGANIZATION Clause	3-22	Computational Phrases	4-32
ACCESS MODE Clause	3-23	INDEX Phrase	4-33
RECORD KEY Clause (Indexed File)	3-24	SIGN Clause	4-35
FILE STATUS Clause	3-26	OCCURS Clause	4-36
I-O-CONTROL Paragraph	3-27	INDICATOR Clause	4-36
RERUN Clause	3-28	SYNCHRONIZED Clause	4-36
SAME Clause	3-28	JUSTIFIED Clause	4-36
MULTIPLE FILE TAPE Clause	3-29	BLANK WHEN ZERO Clause	4-37
COMMITMENT CONTROL Clause	3-29	VALUE Clause	4-38
		General Considerations	4-38
Chapter 4. Data Division	4-1	Format 1 Considerations	4-40
Data Division Concepts	4-1	Format 2 Considerations	4-40
External Data	4-1	PICTURE Clause	4-41
Internal Data	4-2	Symbols Used in the PICTURE	
Data Relationships	4-2	Clause	4-42
Data Division Organization	4-2	PICTURE Clause Editing	4-49
Coding Example	4-3	RENAMES Clause	4-54
Sample Data Division Entries	4-4		
File Section	4-5	Chapter 5. Procedure Division	5-1
Working-Storage Section	4-5	Procedure Division Concepts	5-1
Linkage Section	4-6	Declaratives	5-1
File Description Entry	4-6	Procedures	5-1
File-Name	4-10	Procedure Division Organization	5-2
BLOCK CONTAINS Clause	4-10	Categories of Sentences	5-3
RECORD CONTAINS Clause	4-11	Categories of Statements	5-4
LABEL RECORDS Clause	4-12	Categories of Expressions	5-5
VALUE OF Clause	4-12	Sample Procedure Division Statements	5-6
DATA RECORDS Clause	4-13	Arithmetic Expressions	5-6
LINAGE Clause	4-13	Arithmetic Operators	5-7
CODE-SET Clause	4-16	Conditional Expressions	5-8
Data Description	4-17	Simple Conditions	5-9
Data Description Concepts	4-17	Class Condition	5-9
Level Concepts	4-17	Condition-Name Condition	5-10
Level-Numbers	4-18	Relation Condition	5-11
Special Level-Numbers	4-20	Sign Condition	5-14
Indentation	4-20	Switch-Status Condition	5-14
Classes of Data	4-20	Complex Conditions	5-15
Standard Alignment Rules	4-21	Negated Simple Conditions	5-15
Standard Data Format	4-22	Combined Conditions	5-16
Character-String and Item Size	4-22		

Abbreviated Combined Relation			
Conditions	5-19		
Declaratives	5-20		
EXCEPTION/ERROR Declarative	5-21		
File-Name Phrase	5-21		
INPUT Phrase	5-22		
OUTPUT Phrase	5-22		
I-O Phrase	5-22		
EXTEND Phrase	5-22		
General Considerations	5-22		
Programming Notes	5-23		
Conditional Statements	5-23		
IF Statement	5-24		
Nested IF Statements	5-25		
Input/Output Statements	5-27		
Common Input/Output Phrases	5-27		
Status Key	5-28		
INVALID KEY Condition	5-28		
INTO/FROM Identifier Phrase	5-28		
Current Record Pointer	5-29		
DB-FORMAT-NAME Special Register	5-30		
ACCEPT Statement	5-30		
Format 1 Considerations	5-31		
Format 2 Considerations	5-33		
Format 3 Considerations	5-33		
Format 4 Considerations	5-34		
Programming Notes	5-34		
ACQUIRE Statement	5-34		
CLOSE Statement	5-35		
COMMIT Statement	5-39		
DELETE Statement	5-40		
DISPLAY Statement	5-43		
Format 1 Considerations	5-44		
Format 2 Considerations	5-47		
DROP Statement	5-47		
OPEN Statement	5-47		
READ Statement	5-52		
REWRITE Statement	5-61		
ROLLBACK Statement	5-65		
START Statement	5-66		
WRITE Statement	5-72		
Arithmetic Statements	5-80		
Arithmetic Statement Operands	5-80		
Size of Operands	5-80		
Overlapping Operands	5-81		
Multiple Results	5-81		
Common Phrases	5-82		
CORRESPONDING Phrase	5-82		
GIVING Phrase	5-83		
ROUNDED Phrase	5-83		
SIZE ERROR Phrase	5-84		
ADD Statement	5-84		
COMPUTE Statement	5-85		
DIVIDE Statement	5-86		
MULTIPLY Statement	5-88		
SUBTRACT Statement	5-89		
Data Manipulation Statements	5-90		
INSPECT Statement	5-90		
INSPECT	5-93		
TALLYING Phrase	5-94		
REPLACING Phrase	5-94		
BEFORE/AFTER Phrases	5-95		
INSPECT Statement Examples	5-96		
MOVE Statement	5-97		
General Considerations	5-98		
Elementary Moves	5-98		
Group Moves	5-101		
Format 1 Considerations	5-102		
Format 2 Considerations	5-103		
SET Statement	5-103		
STRING Statement	5-104		
STRING Statement Execution	5-105		
STRING Statement Example	5-106		
UNSTRING Statement	5-109		
Sending Field	5-109		
Data Receiving Fields	5-110		
UNSTRING Statement Execution	5-111		
UNSTRING Statement Example	5-114		
Procedure Branching Statements	5-116		
ALTER Statement	5-116		
Segmentation Information	5-117		
EXIT Statement	5-117		
GO TO Statement	5-118		
Format 1 – Unconditional GO TO	5-119		
Format 2 – Conditional GO TO	5-119		
PERFORM Statement	5-119		
Format 1	5-122		
Format 2	5-122		
Format 3	5-122		
Format 4	5-123		
Varying One Identifier	5-124		
Varying Two Identifiers	5-126		
Varying Three Identifiers	5-130		
Segmentation Information	5-132		
STOP Statement	5-133		
Compiler-Directing Statements	5-134		
ENTER Statement	5-134		
Chapter 6. Additional Functions	6-1		
TABLE HANDLING	6-1		
Table Handling Concepts	6-1		
Table Definition	6-2		
Table References	6-4		
Subscripting	6-4		
Indexing	6-6		
Restrictions on Subscripting and			
Indexing	6-7		
Table Initialization	6-7		

Data Division – Table Handling	6-9	COPY Statement Example	6-49
OCCURS Clause	6-9	SEGMENTATION FEATURE	6-50
Fixed Length Tables	6-11	Segmentation Concepts	6-50
Variable Length Tables	6-11	Program Segments	6-50
ASCENDING/DESCENDING KEY		Fixed Segments	6-50
Phrase	6-12	Independent Segments	6-51
INDEXED BY Phrase	6-13	Segmentation Logic	6-51
USAGE IS INDEX Clause	6-13	Segmentation Control	6-52
Procedure Division – Table Handling	6-14	COBOL Source Program Considerations	6-52
Relation Conditions	6-14	Segmentation – Environment Division	6-52
SEARCH Statement	6-15	Segmentation – Procedure Division	6-53
Format 1	6-17	Segmentation – Special Considerations	6-53
Format 2	6-18	ALTER Statement	6-53
Programming Notes	6-20	PERFORM Statement	6-54
SEARCH Example	6-20	SORT and MERGE Statements	6-54
SET Statement	6-22	Calling and Called Programs	6-54
Format 3	6-23	INTER-PROGRAM COMMUNICATION	
Format 4	6-24	FEATURE	6-54
SORT/MERGE	6-25	Inter-Program Communication Concepts	6-55
Sort/Merge Concepts	6-25	Transfers of Control	6-55
Sort Concepts	6-26	Common Data	6-55
Merge Concepts	6-27	COBOL Language Considerations	6-56
Environment Division – SORT/MERGE	6-27	Data Division – Inter-Program	
File-Control Paragraph	6-27	Communication	6-56
I-O-Control Paragraph	6-27	Record Description Entries	6-58
Data Division – SORT/MERGE	6-28	Data Item Description Entries	6-58
Procedure Division – SORT/MERGE	6-28	Procedure Division – Inter-Program	
MERGE Statement	6-29	Communication	6-58
SORT Statement	6-29	CALL Statement	6-59
MERGE Statement and SORT		USING Phrase	6-60
Statement Phrases	6-30	CPF Graphics Support	6-61
ASCENDING/DESCENDING KEY		CANCEL Statement	6-62
Phrase	6-30	EXIT PROGRAM Statement	6-63
COLLATING SEQUENCE Phrase	6-31	STOP RUN Statement	6-64
USING Phrase	6-32	Inter-Program Communication Feature	
GIVING Phrase	6-32	Examples	6-64
SORT INPUT PROCEDURE Phrase	6-33	DEBUGGING FEATURES	6-66
SORT/MERGE OUTPUT		COBOL Source Language Debugging	6-66
PROCEDURE Phrase	6-34	Compile-Time Switch	6-66
SORT or MERGE INPUT/OUTPUT		Execution-Time Switch	6-67
PROCEDURE Control	6-35	USE FOR DEBUGGING Declarative	6-68
RELEASE Statement (Sort Function		DEBUG-ITEM Special Register	6-70
Only)	6-36	Debugging Lines	6-72
RETURN Statement	6-36	FIPS FLAGGER	6-73
SORT/MERGE Programming Notes	6-37	Chapter 7. TRANSACTION Files (IBM	
SOURCE PROGRAM LIBRARY	6-39	Extension)	7-1
COPY Statement	6-39	Externally Described Transaction File	7-1
Data Field Structures	6-43	Processing an Externally Described	
Floating Point Fields	6-43	TRANSACTION File	7-4
Indicator Structures	6-44	Indicators	7-4
Externally Described COPY		Indicators in the Record Area	7-5
Statement, DDS Format, and DD		Indicators in a Separate Indicator Area	7-5
Format Considerations	6-45		
REPLACING Phrase	6-47		

ASSIGN Clause with Separate Indicator Area Attribute	7-6
Data Description Entry – Boolean Data Special Considerations	7-7
INDICATOR Attribute of the COPY Statement, DDS Format or DD Format	7-7
INDICATORS Phrase	7-9
Indicators in the Record Area	7-9
Indicators in a Separate Indicator Area	7-10
Indicators Sample Programs	7-10
Subfiles	7-22
Use of Subfiles	7-24
Multiple Device Files and Single Device Files	7-28
Program Described Transaction Files ..	7-39
Environment Division	7-39
File-Control Entry	7-39
ASSIGN Clause	7-39
ORGANIZATION Clause	7-40
ACCESS MODE Clause	7-41
RELATIVE KEY Clause	7-41
FILE STATUS Clause	7-41
CONTROL-AREA Clause	7-42
Data Division	7-43
File Description Entry	7-43
Boolean Data Facilities	7-44
Procedure Division	7-44
ACCEPT Statement	7-44
Attribute Data Formats	7-45
ACQUIRE Statement	7-45
CLOSE Statement	7-46
DROP Statement	7-47
OPEN Statement	7-47
Common Processing Facilities	7-48
FORMAT Phrase	7-48
DB-FORMAT-NAME Special Register ..	7-48
INDICATORS Phrase	7-49
SUBFILE Phrase	7-49
TERMINAL Phrase	7-49
READ Statement	7-50
Format 1	7-50
Format 2	7-55
REWRITE Statement	7-58
WRITE Statement	7-60
Format 1	7-60
Format 2	7-63
USE Statement	7-66
Work Station Sample Programs	7-66
Chapter 8. Creating and Executing Programs	8-1
Entering the Source Program into the System	8-1

Using SEU to Enter Source	8-2
Using SEU to Browse through a Compiler Listing	8-4
Compiling the Source Program	8-5
Compiler Options	8-6
Create COBOL Program Command	8-6
PROCESS Statement	8-13
Batch Compiles	8-15
Using COPY within the PROCESS Statement	8-15
Compiler Output	8-16
Command Summary	8-16
Compiler Options	8-17
Source Listing	8-17
Verb Usage by Count Listing	8-20
Data Division Map	8-20
FIPS Messages	8-22
Cross-Reference List	8-23
Messages	8-24
How to Execute a COBOL Program	8-25

Chapter 9. Programmer's Guide

Information	9-1
Device Independence/Device Dependence ..	9-2
Spooling	9-4
Output Spool	9-4
Input Spool	9-4
Level Checking	9-5
File and Record Locking by COBOL	9-5
Releasing a Record Read for Update ...	9-6
Unblocking Input Records and Blocking Output Records	9-6
Multiple Member Processing	9-7
System Override Considerations	9-7
Externally Described/Program Described Files	9-8
Externally Described Files	9-10
Record Format Specifications	9-11
Access Path	9-15
Record Keys and Common Keys ...	9-15
COBOL Specifications for Externally Described Files	9-16
Overriding or Adding COBOL Functions to the External Description	9-18
Program Described Files	9-20
Specific COBOL File Processing	9-21
Printer File Considerations	9-21
SPECIAL-NAMES Paragraph and the ADVANCING Phrase	9-21
LINAGE Clause	9-21
FORMATFILE Files	9-22
Card File Considerations	9-29
Specifying the Function	9-29

Nonassociated Card Files	9-29
Associated Card File	9-30
Stacker Selection	9-30
Hopper Selection	9-30
DISK and DATABASE File	
Considerations	9-31
DATABASE versus DISK Files	9-31
Processing Methods for DISK and DATABASE Files	9-31
Referring to a Partial Key	9-33
Logical File Considerations	9-36
File Processing Methods	9-40
Descending File Considerations	9-41
Commitment Control Considerations	9-44
Exceptions and Some of Their Causes	9-52
System/38 COBOL Programming	
Considerations	9-53
Performance Considerations	9-53
Segmentation	9-53
Debugging	9-53
Data Formats	9-53
*NORANGE Option	9-53
Indicators	9-53
Commitment Control	9-54
Program Loops	9-54
Tracing a Loop in a Program	9-54
Errors That Can Cause a Loop	9-54
Recovery after a Failure	9-55
Recovery with Commitment Control	9-55
Communications Recovery	9-55
System/38 Inter-Program Communication	
Considerations	9-60
Return of Control From a Called Program	9-61
Initialization of Storage	9-61
Local Data Area	9-65
File Considerations	9-65
 Chapter 10. Testing and Debugging	
COBOL Programs	10-1
Using a Test Library	10-1
Testing	10-2
Normal	10-3
Job	10-3
Using Breakpoints	10-4
Example of Using Breakpoints	10-4
Considerations for Using Breakpoints	10-8
Using a Trace	10-8
Example of Using a Trace	10-9
Considerations for Using a Trace	10-11
Using a Debug Execution-Time Switch	10-12
File Status	10-12
Using a COBOL Formatted Dump	10-12
Reply Modes and System Reply List	10-13

Example of Using a Dump	10-13
-------------------------	-------

Chapter 11. COBOL Problem

Determination	11-1
How to Use This Procedure	11-1
Identifying COBOL Problems	11-1
Calling for Help	11-5

Appendix A. COBOL Compiler Service

Information	A-1
Compiler Overview	A-1
Compiler Phases	A-3
Execution-Time Subroutines	A-3
Major Compiler Data Areas	A-3
Compiler Error Message Organization	A-3
Compiler Debugging Options	A-5
DUMP Parameter	A-6
ITDUMP Parameter	A-6
Examples of Using Compiler Debugging Options	A-6
IRP Layout	A-9

Appendix B. Summary of IBM

Extensions	B-1
Character-String Considerations	B-1
Identification Division	B-1
Environment Division	B-1
Data Division	B-2
Procedure Division	B-2
COPY Statement – All Divisions	B-4
TRANSACTION Files	B-4
Compiler Options	B-5

Appendix C. Compile-Time Message

Description	C-1
CAUTION	C-2

Appendix D. Associated Card File

Processing	D-1
Environment Division	D-1
SELECT Clause	D-1
ASSIGN Clause	D-1
Data Division	D-2
Procedure Division	D-2

Appendix E. Intermediate Result

Fields	E-1
Compiler Calculation of Intermediate Results	E-2

Appendix F. Sample File-Processing

Programs	F-1
Sequential File Creation	F-2
Sequential File Updating and Extension	F-4

Indexed File Creation	F-6
Indexed File Updating	F-8
Relative File Creation	F-11
Relative File Updating	F-13
Relative File Retrieval	F-15
Appendix G. COBOL Reserved Words	G-1
Appendix H. EBCDIC and ASCII	
Collating Sequences	H-1
EBCDIC Collating Sequence	H-2
ASCII Collating Sequence	H-5
Appendix I. File Structure Support	
Summary and Status Key Values ...	I-1
Attribute Data Formats	I-10
Display Device Attribute Data	I-10
Communications Device Attribute Data	I-10
OPEN-FEEDBACK and I-O-FEEDBACK	
Data Areas	I-11
OPEN-FEEDBACK	I-11
I-O-FEEDBACK	I-11
Appendix J. Summary of Clauses and Statements	J-1
Conventions Used for Describing Statement Formats	J-1
COBOL Program Structure	J-2

Process Statement	J-2
Identification Division	J-2
Environment Division	J-2
Data Division	J-3
Procedure Division – Format 1 – Declaratives Section	J-3
Procedure Division – Format 2	J-3
DETAILED FORMATS	J-4
Identification Division Format	J-4
Environment Division Formats	J-5
Configuration Section	J-5
Input-Output Section	J-6
Data Division Formats	J-9
File Section Formats	J-9
Working-Storage Section	J-14
Linkage Section	J-16
Procedure Division Formats	J-16
Procedure Division Header	J-16
Procedure Division Statements	J-16
Conditional Expressions	J-37
Qualification of Data Reference Formats	J-39
All Divisions	J-41
Symbols Allowed in the PICTURE Clause	J-43
Assignment-Names in the ASSIGN Clause	J-43
Glossary	GLOSS-1
Index	X-1



About This Manual

Purpose of This Manual

This reference manual describes the System/38 COBOL (Common Business Oriented Language) compiler and language. This manual provides reference material and programmer guide information for persons who have some knowledge of the COBOL language and some experience in writing COBOL programs.

Organization of This Manual

This manual is organized as follows:

- Chapters 1 through 5 describe the COBOL language and each of the four program divisions: Identification, Environment, Data, and Procedure. The COBOL clauses and statements available to the user are explained.
- Chapter 6 describes the additional functions of the language that are provided through the various processing modules.
- Chapter 7 describes the support of work stations and interactive communications as a function of the COBOL compiler.
- Chapter 8 describes how to create and execute programs.
- Chapter 9 contains programmer's guide information.
- Chapter 10 describes how to test and debug COBOL programs.
- Chapter 11 contains COBOL problem determination information.
- Appendix A contains compiler service information.
- Appendix B summarizes IBM extensions.
- Appendix C contains diagnostic level information for compiler messages.
- Appendix D describes associated card file processing.

- Appendix E describes intermediate result fields.
- Appendix F contains sample file-processing programs.
- Appendix G contains COBOL reserved words.
- Appendix H contains the EBCDIC and ASCII collating sequences.
- Appendix I contains a file structure support summary and the status key values.
- Appendix J contains a summary of the System/38 COBOL clauses and statements for each division.
- A glossary of terms and their definitions.

To aid the user, IBM provides several extensions to American National Standard (ANS) COBOL, X3.23-1974. The more significant extensions include:

- TRANSACTION I-O: Used to send or receive records from a work station.
- Data Base I-O: The System/38 COBOL programmer can define data as he does presently. Thus, the COBOL programmer can use standard COBOL Environment and Data Division entries to specify file identification, field definitions, and data structures. Clauses have been added to the READ, WRITE, REWRITE, DELETE, and START verbs to support the System/38 data base.
- COPY: Support for externally described files.
- Extended data types: Computational-3 (packed decimal), and computational-4 (binary).
- Use of apostrophe instead of quotes.

Summary of Changes

The following changes have been made to this manual for COBOL release 8:

- Three references have been added to the list in this prolog: one for Distributed Data Management (DDM), one for the 3180 keyboard template, and one for the office keyboard template.
- A new section on DDM files has been added to Chapter 3.
- A note on when the System/38 compiler does not generate a temporary result field has been added to the "Arithmetic Statements" section of Chapter 5.

- A paragraph on unexpected results during a MOVE operation has been added to Chapter 5.
- A paragraph on replacement has been added to the “COPY Statement” in Chapter 6.
- A description of DDM has been entered into the Glossary.
- The local data area is now defined outside the COBOL program as 1024 bytes of character data.
- Various technical and editorial changes have been made to improve the quality and usability of this manual.

The other updates indicated in this version of the user manual identify corrections to the previous version, or editorial changes made to the text. Note that the way of identifying IBM extensions has changed as described in Chapter 1.

Note: This publication follows the convention that *he* means *he or she*.

What You Should Know

Before reading this manual, you should be familiar with the *IBM System/38 Control Program Facility Concepts Manual*, GC21-7729, which contains the basic concepts of the control program facility.

If You Need More Information

You may need some or all of the following information while using this manual:

- *IBM System/38 Guide to Publications*, GC21-7726, which contains information about related publications, defines terms and lists index entries of frequently used System/38 publications.
- *IBM System/38 Guide to Program Product Installation and Device Configuration*, GC21-7775, which describes how to install COBOL on your system.
- *IBM System/38 COBOL Reference Summary*, SC21-7781, which outlines clauses and statements used in System/38 COBOL.
- *IBM COBOL Coding Form*, GX28-1464, which is used for coding.
- *IBM System/38 Source Entry Utility Reference Manual and User's Guide*, SC21-7722, which describes how to create and update source records using SEU.

- *IBM System/38 Control Program Facility Programmer's Guide*, SC21-7730, which explains how to use CPF commands and data description specifications.
- *IBM System/38 Control Language Reference Manual*, SC21-7731, which describes commands and parameters that are used for various CPF functions.
- *IBM System/38 Control Program Facility Reference Manual – Data Description Specifications*, SC21-7806, which describes the data description specifications that are used for describing files.
- *IBM System/38 Messages Guide: COBOL*, SC21-7823, which contains additional information about COBOL messages.
- *IBM System/38 Concepts for the COBOL User*, GC21-7855, which introduces new extensions to COBOL for System/38.
- *IBM System/38 Problem Determination Guide*, SC21-7876, which contains procedures for resolving system problems that are indicated by error messages, operator/service panel lights, interactive/batch jobs or spooling functions that do not work as expected, or devices that do not work as expected.
- *IBM System/38 Data Communications Programmer's Guide*, SC21-7825, which describes commands, parameters, and data description specification keywords that are used for program-to-program and system-to-device communication functions.
- *IBM System/38 Operator's Guide*, SC21-7735, which explains the operation of individual devices, system operation, and how to analyze system problems.
- *IBM System/38 CPF Graphics Reference Manual*, SC21-8007 and *IBM System/38 CPF Graphics Programmer's Guide*, SC21-8006, which describe how to use the Graphical Data Display Manager (GDDM), and Presentation Graphics Routines (PGR).
- *IBM System/38 Distributed Data Management User's Guide*, SC21-8036.
- *IBM System/38 3180 Keyboard Template*, GX09-1036.
- *IBM System/38 Office Keyboard Template*, GX09-1038.

Industry Standards

The System/38 COBOL compiler is designed according to the following industry standards as understood and interpreted by IBM, as of June 1980:

- The ANS COBOL, X3.23-1974 standard. ANS COBOL is identical to ISO 1989-COBOL, as approved in February 1978 by the International Organization for Standardization. The ANS COBOL processing modules are described in the table under “Language Level” in Chapter 1.
- The December 1975 Federal Information Processing Standard (FIPS PUB 21-1) low-intermediate level. Additional support is provided for many features at higher FIPS levels.

Portions of this manual are copied from American National Standard (ANS) COBOL, X3.23-1974. This material is reproduced with permission from *American National Standard Programming Language COBOL, X3.23-1974*, copyright 1974 by the American National Standards Institute, copies of which can be purchased from the American National Standards Institute at 1430 Broadway, New York, New York, 10018.

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

Acknowledgment

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

Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention COBOL in acknowledgment of the source, but need not quote this entire section.

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

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

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference of Data Systems Languages.

The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the UNIVAC (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

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

Chapter 1. Introduction

General Description

COBOL (Common Business Oriented Language) is a programming language that resembles English. As its name implies, COBOL is especially efficient in the processing of business problems. COBOL can be used to manipulate large files of data in a relatively simple way. That is, COBOL emphasizes the description and handling of data items and of input/output records.

The System/38 COBOL Compiler and Library is an IBM program product that accepts and compiles COBOL programs written in accordance with the ANS COBOL X3.23-1974 standard. This program product also includes a number of IBM extensions. The following sections describe the language level implemented and the language-independent compiler features.

Language Level

The table that follows shows the support of each module provided by System/38 COBOL. The table also describes each module.

The following example explains the notation used to identify levels of implementation:

```
2 SEG 0, 2
|         |         |
|         |         |-----Highest level available
|         |         |-----Minimum level required
|         |         |-----Module identifier
|         |         |-----Level supported by System/38 COBOL
```

System/38 COBOL Processing Modules	Module Description
Nucleus 2 NUC 1, 2	Contains the language elements necessary for internal processing.
Table Handling 2 TBL 1, 2	Contains the language elements necessary for: (1) definition of tables; (2) identification, manipulation, and use of indexes; (3) reference to the items within tables. Provides the ability to define fixed-length or variable-length tables of up to three dimensions. Items in the tables can be referred to by using a subscript or an index.
Sequential I-O 2 SEQ 1, 2	Allows definition and access of sequentially organized external files. IBM System/38 COBOL Sequential I-O provides all level 2 functions except for support of the RERUN clause.
Relative I-O 2 REL 0, 2	Provides the capability of defining and accessing disk files in which records are identified by relative record numbers. A file can be accessed randomly and sequentially in the same COBOL program. IBM System/38 COBOL Relative I-O provides all level 2 functions except for support of the RERUN clause.
Indexed I-O 2 INX 0, 2 <i>Note:</i> Alternate key omitted.	Provides the capability of defining disk files in which records are identified by the value of a key and accessed through an access path. IBM System/38 COBOL Indexed I-O provides all level 2 functions except for support of the ALTERNATE RECORD KEY clause, the RERUN clause, and the KEY IS phrase of the READ statement.
Sort-Merge 2 SRT 0, 2	Allows the inclusion of one or more sorts in a COBOL program and use of the merge facility.
Report Writer 0 RPW 0, 1	Provides semiautomatic production of printed reports.
Segmentation 2 SEG 0, 2	Provides overlaying at object time of Procedure Division sections.
Library 2 LIB 0, 2	Allows inclusion of predefined COBOL text in a program.
Debug 2 DEB 0, 2	Provides for user-specification of statements and procedures for debugging.
Inter-Program Communication 2 IPC 0, 2	Provides facilities for a program to communicate with one or more other programs. Also provides capability to transfer control to another program known at compile time, and the ability for both programs to have access to certain data items.
Communication 0 COM 0, 2	Provides the ability to access, process, and create messages or portions of messages; also provides the ability to communicate through a Message Control System with local and remote communication devices.

Compiler Features

The following language-independent features are made available with System/38 COBOL:

- Syntax-checking compilation saves machine time while debugging source syntax errors. The source program is scanned for syntax errors and associated error messages are generated, but no executable or nonexecutable program is produced.
- The sorted cross-reference option provides a listing of each Data Division name and Procedure Division procedure-name, and indicates the statement numbers of each reference or change to the item.
- Inter-program calls allow programs written in System/38 COBOL to call or be called by other programs written in System/38 COBOL, System/38 RPG III, or System/38 control language.
- Multiple printer files allow the user to define and use multiple printer files in the same program.
- Diagnostic messages below a user-specified level can be suppressed.
- The FIPS flagger, depending on the compiler option chosen, identifies source statements and clauses that do not conform to 1975 FIPS COBOL. 1975 FIPS (Federal Information Processing Standard) COBOL is a compatible subset of 1974 Standard COBOL. A program must conform to one of the four levels of 1975 FIPS COBOL processing: full, high-intermediate, low-intermediate, or low. Refer to the *Messages Guide: COBOL* for information on the messages flagged by FIPS. Figure 1-1 shows the 1974 Standard COBOL processing modules included in each of the levels of 1975 FIPS COBOL.

1974 ANS Module Name	Full FIPS	High- Intermediate FIPS	Low- Intermediate FIPS	Low FIPS
Nucleus	2 NUC 1,2	2 NUC 1,2	1 NUC 1,2	1 NUC 1,2
Table Handling	2 TBL 1,2	2 TBL 1,2	1 TBL 1,2	1 TBL 1,2
Sequential I-O	2 SEQ 1,2	2 SEQ 1,2	1 SEQ 1,2	1 SEQ 1,2
Relative I-O	2 REL 0,2	2 REL 0,2	1 REL 0,2	0 REL 0,2
Indexed I-O	2 INX 0,2	0 INX 0,2	0 INX 0,2	0 INX 0,2
Sort-Merge	2 SRT 0,2	1 SRT 0,2	0 SRT 0,2	0 SRT 0,2
Report Writer	0 RPW 0,1	0 RPW 0,1	0 RPW 0,1	0 RPW 0,1
Segmentation	2 SEG 0,2	1 SEG 0,2	1 SEG 0,2	0 SEG 0,2
Library	2 LIB 0,2	1 LIB 0,2	1 LIB 0,2	0 LIB 0,2
Debug	2 DEB 0,2	2 DEB 0,2	1 DEB 0,2	0 DEB 0,2

Figure 1-1 (Part 1 of 2). 1974 American National Standard and FIPS Levels

1974 ANS Module Name	Full FIPS	High- Intermediate FIPS	Low- Intermediate FIPS	Low FIPS
Inter-Program Communication	2 IPC 0,2	2 IPC 0,2	1 IPC 0,2	0 IPC 0,2
Communications	2 COM 0,2	2 COM 0,2	0 COM 0,2	0 COM 0,2

Key:

```

2 SEG 0, 2
|   |   |
|   |   |-----Highest level available in 1974 ANS COBOL
|   |   |-----Minimum level required by 1974 ANS COBOL
|   |   |-----Module identifier
|   |   |-----Level supported

```

Figure 1-1 (Part 2 of 2). 1974 American National Standard and FIPS Levels

Format Notation

In COBOL, basic formats are prescribed for the various elements of the language. In this manual, these formats are presented in a uniform system of notation that is explained in the following paragraphs. This notation is designed to assist the programmer in writing COBOL source statements.

- Reserved words are printed entirely in CAPITAL LETTERS. These words have preassigned meanings in COBOL. If any reserved word is misspelled, it is not recognized as a reserved word and can cause an error in the program. The two types of reserved words are keywords and optional words.
 - Keywords are required by the syntax of the format unless the portion of the format containing them is optional. In formats, keywords are shown in UNDERLINED CAPITAL LETTERS. A missing keyword is considered an error in the program.
 - Optional words are included only for readability. They can be included or omitted without changing the syntax of the program. Optional words are CAPITALIZED but not underlined.
- Words printed in lowercase letters represent information to be supplied by the user. All such words are defined in the text of this manual.
- For easier text reference, some user-defined words are followed by a hyphen and a digit or letter. This suffix does not change the syntactical definition of the word.
- Braces ({}) enclosing listed items indicate (1) that exactly one of the enclosed stacked items must be specified, and/or (2) when followed by an ellipsis, that the enclosed unit or item must be specified at least once.

- Square brackets ([]) indicate that the enclosed item or unit can be used or omitted, as required for the program. When two or more items are stacked within brackets, one or none of them can be specified. When followed by an ellipsis, the item or unit can be repeated.
- The ellipsis (...) indicates that the immediately preceding unit can occur once or any number of times in succession. A unit can be a single lowercase word or a group of lowercase words and one or more reserved words enclosed in brackets and/or braces. When repetition is used, everything enclosed within the immediately preceding brackets or braces must be repeated.
- The arithmetic and logical operators (+, -, <, >, =) that appear in formats are required although they are not underlined.
- All punctuation and other special characters appearing in formats (except braces, brackets, ellipsis, commas, and semicolons) are required by the syntax of the format when they are shown; if they are omitted, an error occurs in the program. Additional punctuation can be specified, according to the punctuation rules given later in this manual.
- The required clauses and (when written) optional clauses must be written in the sequence shown in the format except where the accompanying text states otherwise.
- Comments, restrictions, and clarifications on the use and meaning of every format are contained in the description following the format.

- IBM extensions to American National Standard COBOL, X3.23-1974, that are part of a command syntax are boxed like this sentence.

- COBOL clauses and statements that are syntax-checked, but are treated as documentation by the System/38 COBOL compiler, are boxed like this sentence.

- IBM extensions

IBM Extension

IBM extensions to American National Standard (ANS) COBOL, X3.23-1974, that are part of the text description begin with the paragraph heading, **IBM Extension** and are separated from the regular text as is this paragraph. Note that Chapter 7 consists only of IBM extensions; the entire chapter is boxed like this paragraph.

End of IBM Extension



Chapter 2. Language Considerations

COBOL Program Structure

Every COBOL source program is divided into four divisions. Each division must begin with a division header and each must be placed in proper sequence. (Appendix J shows the general structure of every COBOL source program.)

The following chapters contain the rules for writing COBOL source programs and the methods of data reference.

The COBOL Divisions

The following paragraphs describe the four divisions of a COBOL source program and their functions in solving a data processing problem.

Identification Division: The Identification Division names the program and, optionally, documents the date the program was written, the compilation date, and other pertinent information.

Environment Division: The Environment Division describes the computer(s) to be used and specifies the machine(s) and equipment features used by the program. This description defines the relationship of data files and input/output devices.

Data Division: The Data Division defines the nature and characteristics of all data the program processes: data used in input/output operations and data developed for internal processing.

Procedure Division: The Procedure Division consists of executable statements that process the user-defined data. Statements are executed in the order they are written unless another order is defined by the user.

Clauses and Statements

Every COBOL source program is written in clauses and statements, each describing a solution to some specific aspect of the data processing problem.

- Clauses, written in the Environment and Data Divisions, specify an attribute of an entry. A series of clauses ending with a period is defined as an entry.
- Statements, written in the Procedure Division, specify an action to be taken by the object program. A series of statements ending with a period is defined as a sentence.

Each clause or statement in the program can be subdivided into smaller syntactical units called phrases. A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL clause or statement. A phrase provides the programmer with required or optional wording, depending on the desired meaning.

Clauses, entries, statements, and sentences can be combined into paragraphs or sections. Each paragraph and section defines some larger part of the data processing problem solution. Specific rules for the formation of each element are given in the documentation for each division of the COBOL program.

Clause and Statement Specification Order

When specified, each required or optional clause and statement (including those treated as documentation) must be written in the sequence shown in the format unless the associated rules explicitly state otherwise. The COBOL program hierarchy for each division is:

- In the Identification, Environment, and Data Divisions:

Division
Section(s)
Entry(s)
Clause(s)
Phrase(s).

- In the Procedure Division:

Division
Section(s)
Sentence(s)
Statement(s)
Phrase(s).

Structure of the Language

In COBOL, the indivisible unit of data is the character. In System/38 COBOL, 51 EBCDIC (extended binary-coded decimal interchange) characters form the COBOL character set: the 26 letters of the alphabet, the 10 Arabic numerals, and 15 special characters.

Individual COBOL characters are put together to form character-strings and separators.

A character-string is a character or sequence of contiguous characters that form a word, a literal, a PICTURE character-string, or a comment. A character-string can be delimited only by a separator.

A separator is a contiguous string of one or more punctuation characters. A separator can be placed next to another separator, or next to a character-string.

Except for comments and nonnumeric literals (which can use any character in the EBCDIC set), the 51 characters are the only characters valid in a COBOL program. Figure 2-1 shows the valid COBOL characters in ascending EBCDIC sequence and their usage in a COBOL program.

IBM Extension

The apostrophe can be used in place of the quotation mark. See "Compiler Options" in Chapter 8.

End of IBM Extension

COBOL Character	Meaning	Use
	Space	Punctuation character
.	Decimal point; period	Editing character; punctuation character
<	Less than	Relation character
(Left parenthesis	Punctuation character
+	Plus symbol	Arithmetic operator; sign; editing character
\$	Dollar sign	Editing character
*	Asterisk	Arithmetic operator; editing character
)	Right parenthesis	Punctuation character
;	Semicolon	Punctuation character

Figure 2-1 (Part 1 of 2). COBOL Characters and Their Meanings

COBOL Character	Meaning	Use
-	Minus symbol; hyphen	Arithmetic operator; sign; editing character
/	Stroke or slash	Arithmetic operator; editing character
,	Comma	Punctuation character; editing character
>	Greater than	Relation character
=	Equal sign	Relation character; punctuation character
" or '	Quotation mark or apostrophe	Punctuation character
A-Z	Alphabet	Alphabetic character
0-9	Arabic numerals	Numeric character

Figure 2-1 (Part 2 of 2). COBOL Characters and Their Meanings

Notes:

1. *Throughout this manual, the quotation mark is used because it is the default option. In all cases, the apostrophe can be used only if the default option is overridden.*
2. *All COBOL characters are considered to be alphanumeric.*

Character-Strings

COBOL character-strings form words, literals, PICTURE character-strings, and comments. Each is described in the following paragraphs.

COBOL Words

A COBOL word can be a user-defined word, a system-name, or a reserved word. A COBOL word can belong to only one of these classes.

The maximum length of a COBOL word is 30 characters.

User-Defined Words: A user-defined word is a COBOL word supplied by the programmer. Valid characters in a user-defined word are:

- A through Z
- 0 through 9
- - (hyphen).

The hyphen cannot appear as the first or last character in a user-defined word.

A list of user-defined word sets, together with rules for their formation, is given in Figure 2-2. The function of each user-defined word in a particular clause or statement is included in the text describing each clause or statement.

User-Defined Word Sets	Rules for Formation
Alphabet-Name Condition-Name Data-Name Record-Name File-Name Index-Name Mnemonic-Name Routine-Name Library-Name Text-Name Program-Name	Must contain at least one alphabetic character. Within each set, the name must be unique either because no other word is made up of an identical character-string, or because it can be made unique through qualification. (See “Methods of Data Reference” later in this chapter.)
Paragraph-Name Section-Name	Need not contain an alphabetic character. Other rules as in first paragraph.
Level-Numbers: 01-49, 66, 77, 88	Must be a 1- or 2-digit integer. Need not be unique.
Segment-Numbers: 00-99	Must be a 1- or 2-digit integer. Need not be unique.

Figure 2-2. User-Defined Word Sets and Rules for Formation

System-Names: A system-name is an IBM-defined name that is used to communicate with the system. A system-name can be:

- A computer-name
- A language-name
- An implementor-name
 - A function-name
 - An assignment-name
 - A user-name.

The function of each system-name is described with the format in which it appears; each system-name is defined in the Glossary.

Reserved Words: A reserved word is a COBOL word with fixed meaning(s) in a COBOL source program. A reserved word must not be specified as a user-defined word or as a system-name. Reserved words can be used only as specified in the formats for a COBOL source program.

Appendix G gives a complete list of COBOL reserved words. “Format Notation” in Chapter 1 gives the conventions used to represent reserved words in this manual.

There are six types of reserved words:

- Keywords

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

Each type is described in the following paragraphs.

Keywords are words that are required within a given clause, entry, or statement. There are three types of keywords:

- Verbs, such as ADD, READ, WRITE
- Required words, which appear in clause, entry, or statement formats, such as the word USING in the MERGE statement
- Words with a specific functional meaning, such as NEGATIVE or SECTION.

Optional Words are words that can be included in a clause, entry, or statement. When an optional word is omitted, the meaning of the COBOL program is unchanged.

There are three types of *connectives*: qualifier, series, and logical.

- Qualifier connectives (OF, IN) associate a data-name, condition-name, text-name, or paragraph-name with its qualifier.
- Series connectives (the comma and semicolon) optionally link two or more consecutive operands. (An operand is a data item or literal that is acted upon by the COBOL program.)
- Logical connectives (AND, OR, AND NOT, OR NOT) are used to specify conditions.

Special registers are compiler-generated storage areas used primarily to store information produced through one of the specific COBOL features. Each such storage area has a fixed name and need not be further defined within the program. These special registers include the following:

IBM Extension

- DB-FORMAT-NAME

(See “Common Input/Output Phrases” under “Input/Output Statements” in Chapter 5.)

End of IBM Extension

- DEBUG-ITEM (see “DEBUGGING FEATURES” in Chapter 6).

- LINAGE-COUNTER (see “LINAGE Clause” in Chapter 4).
- DATE, DAY, TIME (see “ACCEPT Statement” in Chapter 5).

Special-character words are arithmetic operators (+ - * / **) or relation characters (< > =). Arithmetic operators are described under “Arithmetic Expressions” in Chapter 5. Relation characters are described in the relation condition description under “Conditional Expressions” in Chapter 5.

Figurative constants name and refer to specific constant values.

The reserved words and meanings for figurative constants are:

- ZERO, ZEROS, ZEROES: Represents the value 0 or one or more occurrences of the character 0, and is numeric or nonnumeric, depending on context. For example, ZERO is considered to be nonnumeric when it is compared to an alphanumeric data item in a relational expression.
- SPACE, SPACES: Represents one or more blanks or spaces, and is always considered to be a nonnumeric literal.
- HIGH-VALUE, HIGH-VALUES: Represents one or more occurrences of the character with the highest value in the collating sequence used. For the EBCDIC (NATIVE) collating sequence, the character is hex FF; for other collating sequences, the character used depends on the collating sequence. When used in a COBOL program, HIGH-VALUE is treated as a nonnumeric literal.
- LOW-VALUE, LOW-VALUES: Represents one or more occurrences of the character with the lowest value in the collating sequence used. For the EBCDIC (NATIVE) collating sequence, the character is hex 00; for other collating sequences, the character used depends on the collating sequence. When used in a COBOL program, LOW-VALUE is treated as a nonnumeric literal.
- QUOTE, QUOTES: Represents one or more occurrences of the quotation mark character and is nonnumeric. The word QUOTE (QUOTES) cannot be used in place of a quotation mark or an apostrophe to enclose a nonnumeric literal.

IBM Extension

When APOST is specified as a compiler option, the figurative constant QUOTE has the EBCDIC value of an apostrophe.

End of IBM Extension

- ALL literal: Represents one or more occurrences of the string of characters composing the literal and is nonnumeric. The literal must be either a nonnumeric literal, Boolean literal, or a figurative constant other than the ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only. The figurative

constant, ALL literal, cannot be used with the DISPLAY, INSPECT, STRING, STOP, or UNSTRING statements.

The singular and plural forms of a figurative constant are equivalent and can be used interchangeably. For example, if DATA-NAME-1 is a five-character data item, either of the following statements will fill DATA-NAME-1 with five spaces:

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

In any format, a figurative constant can be substituted for a nonnumeric literal; only the figurative constant ZERO (ZEROS, ZEROES) can be substituted for a numeric literal.

IBM Extension

The figurative constant ZERO can be used as a Boolean literal.

End of IBM Extension

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

- When a figurative constant is associated with a data item, the length of the figurative constant character-string is equal to the length of the associated data item. This rule applies, for example, when a figurative constant is moved to, or compared with, another item.
- When a figurative constant is not associated with another data item, the length of the character-string is one character. This rule applies, for example, in the DISPLAY, INSPECT, STRING, STOP, and UNSTRING statements.

Literals

A literal is a character-string whose value is specified either by the ordered set of characters of which it is composed, or by specification of a figurative constant. The three types of literals are Boolean, nonnumeric, and numeric.

IBM Extension

Boolean Literals: A Boolean literal is a character-string delimited on the left by the separator B" and on the right by the quotation mark separator. The character-string consists only of the character 0 or 1. The value of a Boolean literal is the character itself, excluding the delimiting separators. All Boolean literals are of the category Boolean.

End of IBM Extension

Nonnumeric Literals: A nonnumeric literal is a character-string that can contain any allowable character from the EBCDIC set.

IBM Extension

A nonnumeric literal can contain a maximum of 160 characters.

End of IBM Extension

A nonnumeric literal must be enclosed by quotation marks (or apostrophes, if the APOST option is in effect). The enclosing quotation marks are not part of the literal.

Any punctuation characters included within a nonnumeric literal are part of the value of the literal. An embedded quotation mark must be represented by two adjacent quotation marks (" "); one quotation mark (") is then part of the value of the literal.

Every nonnumeric literal is in the alphanumeric data category. Data categories are defined under "PICTURE Clause" in Chapter 4.

Numeric Literals: A numeric literal is a character-string whose characters are selected from the digits 0 through 9, the sign characters (+ or -), and the decimal point. The following rules apply:

- 1 to 18 digits are allowed.
- Only one sign character is allowed. If a sign character is included, it must be the leftmost character of the literal. If the literal is unsigned, it is considered to have a positive value.
- Only one decimal point is allowed. If a decimal point is included, it is treated as an assumed decimal point (not considered a character position in the literal). The decimal point can appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, it is considered to be an integer. The word *integer* appearing in a format represents a numeric literal of nonzero value that contains no sign and no decimal point; any other restrictions are included with the description of the format.

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

If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a nonnumeric literal.

PICTURE Character-Strings

A PICTURE character-string consists of COBOL characters used as symbols in the PICTURE clause.

The choice of symbols determines whether the user-supplied name is numeric, alphabetic, or alphanumeric. The character-string can also be used to define edited output fields.

Comments

A comment is a character-string containing any combination of characters from the EBCDIC set. A comment serves only as documentation. Comments take two forms:

- A comment entry in the Identification Division: For a further description of a comment entry, see "IDENTIFICATION DIVISION" in Chapter 3.
- A comment line (preceded by an asterisk or a slash in Column 7) in any division of the program: For a further description of a comment line, see "Standard COBOL Format" later in this chapter.

Separators

A separator is a string of one or more punctuation characters or B" when used to delimit a Boolean literal. The characters are shown in Figure 2-3.

Punctuation Character	Meaning
	Space
	Period
(Left parenthesis
)	Right parenthesis
;	Semicolon
,	Comma
=	Equal sign
"	Quotation mark
'	Apostrophe
B"	Delimiter for Boolean literal

Figure 2-3. Punctuation Characters

The following rules apply to the formation of separators:

- A space is always a separator except when the space appears within a nonnumeric literal. When contained between the opening and closing quotation marks of a nonnumeric literal, the space is considered part of the literal. Wherever a space is used as a separator, more than one space can be used.

- A comma, semicolon, or period immediately followed by a space is a separator. These separators can appear only where explicitly allowed by COBOL rules.
- The left and right parentheses are separators. Parentheses must appear as balanced pairs of left and right parentheses to delimit subscripts, indexes, arithmetic expressions, or conditions.
- The quotation mark is a separator. An opening quotation mark must be immediately preceded by a space or a left parenthesis. A closing quotation mark must be immediately followed by one of the following separators: space, comma, semicolon, period, or right parenthesis. Quotation marks must appear as balanced pairs delimiting nonnumeric literals except when the literal is continued.
- The pseudo-text delimiter (= =) is a separator. 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 following separators: space, comma, semicolon, or period. Pseudo-text delimiters must appear as balanced pairs delimiting pseudo-text.

IBM Extension

- B" is a separator when it is used to describe a Boolean literal. The B must immediately precede the quotation mark.

End of IBM Extension

Standard COBOL Format

COBOL programs must be written in the standard COBOL format, described in the following discussion. The format is described in terms of an 80-character line. The output listing of the source program is printed in this same format. The COBOL coding form is shown in Figure 2-4.

Sequence Numbers (Columns 1-6)

Sequence numbers are written in columns 1 through 6. A sequence number numerically identifies each line to be compiled by the COBOL compiler. Sequence numbers are optional. A sequence number, if used, must consist of six digits in the sequence number area, (including the preprinted digits in columns 4 and 5).

If sequence numbers are used in the source program, they must be in ascending order. If sequence numbers are out of sequence, the compiler accepts them in the order read and generates a warning message.

IBM Extension

The user can suppress sequence checking at compile time by specifying NOSEQUENCE.

If the NUMBER option is specified, the sequence numbers from columns 1 through 6 are used; otherwise the source sequence numbers provided in the source file are used.

_____ End of IBM Extension _____

IBM		COBOL Coding Form										PAGE	OF										
SYSTEM		PUNCHING INSTRUCTIONS																					
PROGRAM		GRAPHIC								CARD FORM #		*											
PROGRAMMER		DATE		PUNCH																			
SEQUENCE		A							B	COBOL STATEMENT												IDENTIFICATION	
(PAGE)	SERIAL																						
1	4																						
	01																						
	02																						
	03																						
	04																						

Columns 1-6 represent the sequence number area.
 Column 7 is the continuation area.
 Columns 8-11 represent Area A } Used for writing COBOL source statements.
 Columns 12-72 represent Area B }
 Columns 73-80 are used to identify the program.

Figure 2-4. IBM COBOL Coding Form and Standard COBOL Format

Continuation Area (Column 7)

The continuation area is used to indicate the continuation of words and nonnumeric literals from the previous line onto the current line, to specify debugging lines, or to indicate that the text on this line is to be treated as a comment.

Area A (Columns 8-11) and Area B (Columns 12-72)

COBOL elements that can begin in Area A and specific COBOL elements that can follow them are shown in Figure 2-5.

The basic skeleton of a COBOL program is shown in Figure 2-6.

Elements That Must Begin in Area A	Must Be Followed Immediately By	Placement of Following Elements
Division header	(In Procedure Division) USING phrase Section header, paragraph header, paragraph-name, or (in Procedure Division) keyword DECLARATIVES	Same or next line (Area B) Next line (Area A)
Section header	(In Declaratives section) USE statement Paragraph header, paragraph-name, (after USE, if specified), level indicator, or level-number	Same or next line (Area B) Next line (Area A)
Paragraph header or paragraph-name	Identification Division entry, Environment Division entry, or Procedure Division sentence	Same or next line (Area B)
Level indicator, level-numbers 01 and 77	Data-name	Same or next line (Area B)
Keyword DECLARATIVES	Declaratives section-name	Next line (Area A)
Keywords END DECLARATIVES	Section header	Next line (Area A)

Figure 2-5. Sequence of Elements in Area A and Area B

SEQUENCE	(PAGE)	SERIAL	COBOL	A	B
1	3	4	6	7	8
001	0	1	0	IDENTIFICATION DIVISION.	
002	0	2	0		
003	0	3	0	ENVIRONMENT DIVISION.	
004	0	4	0	CONFIGURATION SECTION.	
005	0	5	0		
006	0	6	0	INPUT-OUTPUT SECTION.	
007	0	7	0	FILE-CONTROL.	
008	0	8	0		
009	0	9	0	DATA DIVISION.	
010	0	10	0	FILE SECTION.	
011	0	11	0	FD	→
012	0	12	0	WORKING-STORAGE SECTION.	
013	0	13	0	FILE DESCRIPTION	→
014	0	14	0	FILE	→
015	0	15	0		
016	0	16	0	PROCEDURE DIVISION.	
017	0	17	0	DECLARATIVES.	
018	0	18	0		
019	0	19	0	END DECLARATIVES.	
020	0	20	0	SECTION-NAME SECTION.	
			0	PARAGRAPH-NAME	
			0	COMMENTS.	
			0	DEBUG-STATEMENTS.	
			0	MOVE "TEST" TO A.	

Figure 2-6. Basic Skeleton of a COBOL Program

Special Considerations

Some lines in a COBOL program require additional rules. A discussion of each follows.

Division Header

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

Section Header

A section header must be immediately followed by a period except when Procedure Division segment numbers are specified. In the Environment and Procedure Divisions, a section consists of paragraphs. In the Data Division, a section consists of Data Division entries.

Paragraph Header, Paragraph-Name

In both the Identification Division and the Environment Division, a paragraph consists of a paragraph header followed by one or more entries in Area B. An entry consists of one or more clauses. In the Procedure Division, a paragraph consists of a paragraph-name followed by one or more sentences in Area B. A sentence consists of one or more statements; a statement is a syntactically valid combination of a COBOL verb and its operands. Entries and sentences must be ended with a period followed by a space.

Successive entries or sentences begin in Area B. The entries are either on the same line as the last entry or sentence, or they are on the next succeeding nonblank noncomment line.

Data Division Entries

Each Data Division entry begins with a level indicator or level-number followed by a space. On the same line is a data-name in Area B, followed by a sequence of independent clauses describing the item. Each clause, except the last, is followed by a space (or optionally by a comma or semicolon and a space). The last clause in the entry must be ended with a period followed by a space.

Successive clauses begin in Area B. The clauses are either on the same line as the preceding clause, or on the next succeeding nonblank noncomment line.

A level indicator (FD, SD) must begin in Area A and be followed by a space. For a further description of level indicators, see "Data Division Organization" in Chapter 4.

A level-number is a 1- or 2-digit integer with one of the following values: 1 through 49, 66, 77, or 88. At least one space must follow the level-number.

Level-numbers 01 and 77 must begin in Area A. The associated record-name or item-name must appear in Area B. Level-numbers 02 through 49, 66, and 88 can begin in either Area A or Area B.

DECLARATIVES and END DECLARATIVES

In the Procedure Division, the keywords `DECLARATIVES` and `END DECLARATIVES` begin and end the Declaratives portion of the source program. Both of these keywords must begin in Area A and be followed immediately by a period. No other text can appear on the same line. After the keyword `END DECLARATIVES`, no text can appear before the following section header.

Program Spacing

In writing a COBOL program, rules for indentation, continued lines, comment lines, debugging lines, and blank lines must be observed.

Indentation

Within an entry or sentence, successive lines in Area B can have the same format or can be indented to clarify program logic. The output listing is indented only if the input statements are indented. Indentation does not affect the syntax of the program. The amount of indentation can be chosen by the user, subject only to the restrictions on the width of Area B.

Continuation of Lines

Any sentence, entry, clause, or phrase that requires more than one line can be continued in Area B of the next succeeding noncomment line. The line being continued is called the continued line; the succeeding lines are continuation lines. Area A of a continuation line must contain only spaces.

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

If there is a hyphen in the continuation area of a line, the first nonblank character of this continuation line immediately follows the last nonblank character of the continued line without any intervening space. However, this restriction does not apply to nonnumeric literals.

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

A pair of quotation marks indicating a single quotation mark in the value of the literal must occur on the same line. Likewise, both characters composing the separator `==` or `B"` must be on the same line.

Comment Lines

A comment line is any line with an asterisk or slash in the continuation area of the line. The comment may be written anywhere in Area A and Area B of that line. The comment may consist of any combination of characters from the EBCDIC set.

If an asterisk is placed in the continuation area, this comment line is printed in the output listing immediately following the last preceding line.

If the slash is placed in the continuation area, the current page of the output listing is ejected, and the comment line is printed on the first line of the next page.

The asterisk or slash and the comment are produced only on the output listing. They are treated as documentation by the compiler.

Successive comment lines are allowed. Each must begin with an asterisk or slash in the continuation area.

Comment lines are not allowed before the Identification Division header.

Debugging Lines

A debugging line is any line with a D coded in the continuation area. Rules for the formation of debugging lines are given under "DEBUGGING FEATURES" in Chapter 6.

Blank Lines

Blank lines contain nothing but spaces from Column 7 through Column 72. A blank line may appear anywhere in a program except immediately preceding a continuation line.

Overall Punctuation Rules

Any punctuation character included in a PICTURE character-string, a comment character-string, or a nonnumeric literal is not considered to be a punctuation character but rather is considered to be part of the character-string or literal.

A comma, period, or semicolon followed by a space in or at the end of a PICTURE character-string is a separator and terminates the PICTURE character-string. The comma and semicolon are used only for readability.

Punctuation rules for each division of the COBOL source program follow.

Identification Division

Commas and semicolons can be used in the comment-entries. The PROGRAM-ID paragraph must end with a period followed by a space.

Environment Division

Commas or semicolons can separate successive clauses and successive operands within clauses. The SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, and I-O-CONTROL paragraphs must each end with a period followed by a space. In the FILE-CONTROL paragraph, each file-control entry must end with a period followed by a space.

Data Division

Commas or semicolons may separate successive clauses and operands within clauses. File (FD), Sort/Merge file (SD), and data description entries must each end with a period followed by a space.

Procedure Division

Commas or semicolons may separate successive statements within a sentence and successive operands within a statement. Each sentence and each procedure must end with a period followed by a space.

Methods of Data Reference

Every user-specified name defining an element in a COBOL program must be unique, either because no other name has a character-string of the same value or because it can be made unique through qualification, subscripting, or indexing. In addition, references to data and procedures can be either explicit or implicit. The rules for qualification and for explicit and implicit references follow.

Qualification

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

Qualification is specified by placing one or more phrases after a user-specified name. Each phrase consists of the word OF or IN followed by a qualifier. (OF and IN are logically equivalent.) The three formats for references are references to Data Division names, references to Procedure Division names, and references to COPY libraries.

Format 1

$$\left\{ \begin{array}{l} \text{data-name-1} \\ \text{condition-name} \end{array} \right\} \left[\left[\begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right] \text{data-name-2} \right] \dots$$

Format 2

paragraph-name [{ OF } section-name]

Format 3

text-name [{ OF } library-name]

In Data Division references, all qualifying data-names must be associated with a level indicator or level-number. Therefore, two identical data-names must not appear as subordinate entries in a group item unless they can be made unique through qualification. Names associated with a level indicator (FD and SD) are the highest level in the hierarchy. Next highest are those associated with level-number 01. Names associated with level-numbers 02 through 49 are at successively lower levels in the hierarchy.

In the Procedure Division, two identical paragraph-names must not appear in the same section. A section-name is the highest and only qualifier available for a paragraph-name.

The following example illustrates the use of identical names in a section hierarchy:

```
01 FIELD-A
  02 FIELD-B
    05 SUB1
    07 SUB2
  02 FIELD-C
    07 SUB1
```

A hierarchy includes all subordinate entries to the next equal or higher level-number. Therefore, in the above example all entries are in the hierarchy of FIELD-A. All entries from FIELD-B to, but not including, FIELD-C are in the hierarchy of FIELD-B.

In the hierarchy of FIELD-A, SUB1 can be used twice; once as subordinate to FIELD-B and once as subordinate to FIELD-C. In references to SUB-1, it must be qualified as SUB-1 OF FIELD-B or SUB-1 OF FIELD-C. Within FIELD-B or FIELD-C, SUB1 cannot be subordinate to itself.

In any hierarchy, the name associated with the highest level must be unique and cannot be qualified.

No matter what qualification is available, no name can be both a data-name and a procedure-name.

Enough qualification must be specified to make the name unique; however, it may not be necessary to specify all the levels of the hierarchy. For

example, if more than one file has records that contain the field EMPLOYEE-NO but only one of the files has a record named MASTER-RECORD, then specifying EMPLOYEE-NO OF MASTER-RECORD sufficiently qualifies EMPLOYEE-NO. EMPLOYEE-NO OF MASTER-RECORD OF MASTER-FILE is valid but unnecessary.

Qualification Rules

The following rules for qualification apply:

- Each qualifier must be of a successively higher level and must be within the same hierarchy as the name it qualifies.
- The same name must not appear at two levels in a hierarchy unless it can be qualified.
- If a data-name or condition-name is assigned to more than one data item, the data item must be qualified each time it is referenced, with this exception: in the REDEFINES clause, qualification is unnecessary and must not be used.
- A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to within the section in which it appears.
- Library-name must be unique in the system. Therefore, the first 10 characters of library-name must be unique.
- Text-name (member-name) must be qualified by the file-name-library-name in which it resides. If no library is specified, the *LIBL list is searched.

IBM Extension

File-name is optional for the COPY statement, format 1. If file-name is not specified, the default is QCBLSRC.

End of IBM Extension

- When a data-name is being used as a qualifier, it cannot be subscripted.
- A name can be qualified even when it does not need qualification.
- If more than one combination of qualifiers ensures uniqueness, then any of these combinations can be used.
- Duplicate section-names are not allowed.
- A data-name cannot be the same as a section-name or a paragraph-name.
- A section-name cannot be the same as a paragraph-name.

- If a data-name cannot be made unique by qualification, duplication of this data-name is not allowed.
- The complete list of qualifiers for one data-name must not be the same as a partial list of qualifiers for another data-name.
- A maximum of 48 qualifiers (49 qualifiers for file data) can be specified.
- LINAGE-COUNTER must be qualified each time it is referenced if more than one file description entry containing a LINAGE clause has been specified in the source program.



Subscripting and Indexing

Subscripts and indexes can be used only when reference is made to an individual element within a table of elements that have not been assigned individual data-names. Subscripting and indexing are explained under “TABLE HANDLING” in Chapter 6.

Identifier

An identifier is a term used to reflect that a data-name, if not unique in a program, must be followed by some syntactically correct combination of qualifiers, subscripts, or indexes sufficient to ensure uniqueness. The general formats for identifiers are as follows:

Format 1

$$\text{data-name-1} \left[\left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\} \text{data-name-2} \right] \dots \left[\left(\text{subscript-1} \left[, \text{subscript-2} \left[, \text{subscript-3} \right] \right] \right) \right]$$


Format 2

$$\text{data-name-1} \left[\left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\} \text{data-name-2} \right] \dots \left[\left(\left\{ \begin{array}{c} \text{index-name-1} \left[\left\{ \pm \right\} \text{literal-2} \right] \\ \text{literal-1} \end{array} \right\} \right. \right. \\ \left. \left. \left[, \left\{ \begin{array}{c} \text{index-name-2} \left[\left\{ \pm \right\} \text{literal-4} \right] \\ \text{literal-3} \end{array} \right\} \right] \left[, \left\{ \begin{array}{c} \text{index-name-3} \left[\left\{ \pm \right\} \text{literal-6} \right] \\ \text{literal-5} \end{array} \right\} \right] \right] \right) \right]$$

Restrictions on qualification, subscripting, and indexing follow:

- A data-name must not be subscripted or indexed when that data-name is being used as an index, subscript or qualifier.



- Indexing is not permitted when subscripting is not permitted.
- An index can be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit the values associated with index-names to be stored as a binary occurrence number. Such data items are called index data items.
- Literal-1, literal-3, literal-5 in the above format must be positive numeric integers. Literal-2, literal-4, literal-6 must be unsigned numeric integers.

Condition-Name

A condition-name is a user-defined word that is assigned a specific value or range of values. The value assigned is contained in the set of values that a conditional variable may possess. A condition-name can alternatively be a user-defined word that is assigned the status of an IBM-defined switch or device.

Each condition-name must be unique, or it must be made unique through qualification, and/or indexing, or subscripting.

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

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

The format and restrictions on the combined use of qualification, subscripting, and indexing of condition-names are the same as that for identifiers except that data-name-1 is replaced by condition-name-1.

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

Explicit and Implicit References

COBOL source program references can be either explicit or implicit in three instances: data attribute specification, Procedure Division data references, and transfers of control.

Data Attribute Specification

Explicit attributes are specified in COBOL coding. If a data attribute is not specified in COBOL coding, it takes on a default value. These default values are implicit attributes.

For example, the ACCESS MODE clause in the file-control entry need not be specified. If the clause is omitted, the compiler provides the default value, ACCESS MODE IS SEQUENTIAL. This clause is then an implicit attribute. If

this same attribute, ACCESS MODE IS SEQUENTIAL, is specified in the COBOL coding, it is an explicit attribute.

Procedure Division Data References

Procedure Division statements can refer to data items either explicitly or implicitly.

An explicit reference occurs when the data-name of the item is written in a COBOL statement or when the data-name is copied into the program through a COPY statement. An implicit reference occurs when the data-name is referred to by a COBOL statement without the name being written in that statement.

For example, when USE AFTER STANDARD EXCEPTION/ERROR PROCEDURE ON INPUT is specified, an implicit reference is made to each file-name that identifies an input file. See “EXCEPTION/ERROR Declarative” in Chapter 5 for a further description.

Transfers of Control

In the Procedure Division, program flow transfers control from statement to statement in the order they are written unless an explicit control transfer is specified or no next executable statement exists. (See the note below.) This normal program flow is an implicit transfer of control.

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

- After execution of the last statement of a procedure being executed under control of another COBOL statement. COBOL statements that control procedure execution are MERGE, PERFORM, SORT, and USE.
- During SORT or MERGE statement execution when control is transferred to any input or output procedure.
- During execution of any COBOL statement that causes execution of a Declarative procedure.
- At the end of execution of any Declarative procedure.

COBOL also provides explicit transfers of control through the execution of a procedure branching or conditional statement. Lists of procedure branching and conditional statements are given under “Procedure Division Organization” in Chapter 5.

Note: The term *next executable statement* refers to the next COBOL statement to which control is transferred according to the rules given above. No next executable statement can follow:

- The last statement in a Declarative procedure that is not being executed under control of another COBOL statement.
- The last statement in a COBOL program when the paragraph in which it appears is not being executed under control of another COBOL statement.



Chapter 3. Identification and Environment Divisions

IDENTIFICATION DIVISION

The Identification Division must be the first division in every COBOL source program. This division names the source program and the object program. (A source program is the user-written COBOL program. An object program is the output from a compilation.)

The user may also include the date the program was written, the date of compilation, and other such documentary information about the program in the Identification Division.

Format

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. program-name.  
[ AUTHOR. [comment-entry] . . . ]  
[ INSTALLATION. [comment-entry] . . . ]  
[ DATE-WRITTEN. [comment-entry] . . . ]  
[ DATE-COMPILED. [comment-entry] . . . ]  
[ SECURITY. [comment-entry] . . . ]
```

The Identification Division must begin with the words IDENTIFICATION DIVISION followed by a period and a space.

Coding Example

SEQUENCE	CONT	A	B
(PAGE)	SERIAL		
1	3	4	6
7	8	12	16
20	24	28	32
36			
001	01	O IDENTIFICATION DIVISION.	
	02	C PROGRAM-ID. SAMPLE.	
	03	O AUTHOR. A PROGRAMMER.	
	04	C INSTALLATION. ROCHESTER LAB.	
	05	O DATE-WRITTEN. 04/11/79.	
	06	O DATE-COMPILED. 08/16/79.	
	07	O SECURITY. NON-CONFIDENTIAL.	
	08		

PROGRAM-ID Paragraph

The first paragraph of the Identification Division must be the PROGRAM-ID paragraph. The PROGRAM-ID paragraph specifies the name by which the program is known to the system.

The name by which the program is known to the system can be overridden by the PGM parameter of the CRTCLPGM command. See Chapter 8 for more information on the PGM parameter.

Program-name is a user-defined word that identifies the object program to the system. A program-name must include at least one alphabetic character. The system uses the first 10 characters of program-name as the identifying name of the program; these first 10 characters, therefore, should be a unique program-name.

The system expects the first character of program-name to be alphabetic; if it is numeric, it is converted as follows:

- 0 is converted to J
- 1 through 9 is converted to A through I.

The system does not include the hyphen as an allowable character; therefore, if any of the second through tenth characters are hyphens, they are converted to zeros.

To avoid such conversions, the user should not specify program-names with leading numerics or embedded hyphens.

Other Optional Paragraphs

The other paragraphs are optional; however, if they are written, they must appear in the order shown in the format.

The comment-entries serve only as documentation and do not affect the syntax of the program. The comment-entries in the optional paragraphs may be any combination of characters from the EBCDIC set and may be written in Area B on one or more lines. A hyphen is not permitted in the continuation area of Identification Division statements.

The DATE-COMPILED paragraph provides the compilation date of the source listing. When the comment-entry is specified, the entire entry is replaced with the current date. When the comment-entry is omitted, the compiler adds the current date to the DATE-COMPILED paragraph.

ENVIRONMENT DIVISION

The Environment Division, the second division of all COBOL source programs, identifies the following:

- The computer on which the source program is to be compiled
- The computer on which the object program is to be executed
- The specific main storage size required to execute the object program
- The linkage between the logical concept of the files and their records, and the physical aspects of the devices on which data is stored.

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

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

Format

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. source-computer-entry
OBJECT-COMPUTER. object-computer-entry
 [SPECIAL-NAMES. special-names-entry]
 [INPUT-OUTPUT SECTION.
FILE-CONTROL. {file-control-entry} . . .
 [I-O-CONTROL. input-output-control-entry]]

The Environment Division must begin with the words ENVIRONMENT DIVISION followed by a period and a space.

Coding Example

SEQUENCE		PAGE		SERIAL		A		B				
1	3	4	6	7	8	12	16	20	24	28	32	36
00	2	0	1	0		ENVIRONMENT	DIVISION.					
	0	2	0			CONFIGURATION	SECTION.					
	0	3	0			SOURCE-COMPUTER.	IBM-S38.					
	0	4	0			OBJECT-COMPUTER.	IBM-S38.					
	0	5	0			SPECIAL-NAMES.	C01 IS PAGE-TOP.					
	0	6	0			INPUT-OUTPUT	SECTION.					
	0	7	0			FILE-CONTROL.						
	0	8	0			SELECT						
	0	9	0			ORGANIZATION						
	1	0										

Configuration Section

The Configuration Section describes the computer that compiles the source program and the computer that executes the object program. This section optionally relates IBM-defined function names to user-defined mnemonic-names, specifies the collating sequence to be used, specifies a substitution for the currency sign, and/or interchanges the functions of the comma and the period.

In the Configuration Section, the comma or semicolon can optionally separate successive clauses within a paragraph. In each paragraph, there must be one period; the period must be placed immediately after the last entry in the paragraph.

Format

CONFIGURATION SECTION.

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

OBJECT-COMPUTER. computer-name

[, MEMORY SIZE integer { WORDS
CHARACTERS
MODULES }]

[, PROGRAM COLLATING SEQUENCE IS alphabet-name]

[, SEGMENT-LIMIT IS segment-number] .

[SPECIAL-NAMES. [function-name-1 IS mnemonic-name] . . .

[function-name-2

{ IS mnemonic-name , ON STATUS IS condition-name-1 [, OFF STATUS IS condition-name-2]
IS mnemonic-name , OFF STATUS IS condition-name-2 [, ON STATUS IS condition-name-1]
ON STATUS IS condition-name-1 [, OFF STATUS IS condition-name-2]
OFF STATUS IS condition-name-2 [, ON STATUS IS condition-name-1] } . . .

[, alphabet-name IS { STANDARD-1
NATIVE
literal-1 [{ THROUGH
THRU } literal-2
ALSO literal-3 [, ALSO literal-4] . . .]
[literal-5 [{ THROUGH
THRU } literal-6
ALSO literal-7 [, ALSO literal-8] . . .]] . . . } . . .

[, CURRENCY SIGN IS literal-9]

[, DECIMAL-POINT IS COMMA] .]

SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph describes the computer that compiles the source program. The computer name should be coded as: IBM-S38.

With the exception of the WITH DEBUGGING MODE clause, the SOURCE-COMPUTER paragraph is syntax-checked, but is treated as documentation. The WITH DEBUGGING MODE clause is described under “DEBUGGING FEATURES” in Chapter 6.

OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph identifies the computer that executes the object program. Computer-name must be the first entry in the OBJECT-COMPUTER paragraph. The other clauses can be specified in any order. The computer-name should be coded as: IBM-S38.

MEMORY SIZE Clause

The MEMORY SIZE clause is syntax-checked, but is treated as documentation.

PROGRAM COLLATING SEQUENCE Clause

The PROGRAM COLLATING SEQUENCE clause specifies the collating sequence used in a program. The collating sequence associated with the specified alphabet-name must be defined in the SPECIAL-NAMES paragraph. The program collating sequence applies to the following nonnumeric comparisons:

- Those comparisons explicitly specified in IF, PERFORM, and SEARCH statements
- Those comparisons implicitly specified in STRING, INSPECT, and UNSTRING statements
- Those comparisons implicitly specified in MERGE or SORT statements that do not specify a COLLATING SEQUENCE phrase.

When the PROGRAM COLLATING SEQUENCE clause is omitted, the EBCDIC collating sequence is used. See Appendix H for the complete EBCDIC collating sequence.

SEGMENT-LIMIT Clause

The SEGMENT-LIMIT clause is described under “SEGMENTATION FEATURE” in Chapter 6.

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph relates IBM-specified function-names to user-specified mnemonic-names. This paragraph specifies a collating sequence that is associated with an alphabet-name, a substitute character for the currency sign, and the interchange of the comma and decimal point in PICTURE clauses and numeric literals. The clauses can be specified in any order.

Function-Name-1 Clause

Function-name-1 specifies system devices or standard system actions taken by the compiler.

The associated mnemonic-name is required. The mnemonic-name is formed according to the rules for a user-defined word and must contain at least one alphabetic character.

Programming Note: The SEU Syntax Checker requires that the first clause be entered on the same line as the SPECIAL-NAMES entry.

Figure 3-1 shows the actions that are associated with mnemonic-names for function-name-1. Each of these functions can appear only once in the SPECIAL-NAMES paragraph.

Function-name-1	Statement where mnemonic-name associated with function-name is used	Usage
CSP	WRITE	Suppress spacing when printing a line. Use only when PRINTER is the device. See "FILE-CONTROL Paragraph" later in this chapter.
C01	WRITE	Skip to the next page. Use only when PRINTER is the device. See "FILE-CONTROL Paragraph" later in this chapter.
S01, S02, S03, S04, S05	WRITE	Select stackers on a card punch file. S01 through S04 select stackers 1 through 4, and S05 selects stacker 1 on the IBM 5424. Use only when PUNCH, PUNCHPRINT, or PRINT is the device. See "FILE-CONTROL Paragraph" later in this chapter.
ATTRIBUTE-DATA	ACCEPT	Retrieve attribute data about a program device acquired by a TRANSACTION file, but only when the file is open. See "ACCEPT Statement" in Chapter 7.

Figure 3-1 (Part 1 of 2). Choices of Function-Name-1 and Action Taken

Function-name-1	Statement where mnemonic-name associated with function-name is used	Usage
I-O-FEEDBACK	ACCEPT	Give information about the last I-O operation on a file, but only when the file is open. See "ACCEPT Statement" in Chapter 5.
OPEN-FEEDBACK	ACCEPT	Give information about a file, but only when the file is open. See "ACCEPT Statement" in Chapter 5.
CONSOLE, SYSTEM-CONSOLE	ACCEPT, DISPLAY	Communicate with the system operator's message queue (QSYSOPR).
LOCAL-DATA	ACCEPT, DISPLAY	Retrieve data from, or moves data to the local data area created by the system for every job. See "ACCEPT Statement" and "DISPLAY Statement" in Chapter 5.
REQUESTOR	ACCEPT, DISPLAY	Communicate with the user work station (interactive jobs) or the batch input stream or job log (batch jobs).

Figure 3-1 (Part 2 of 2). Choices of Function-Name-1 and Action Taken

Function-Name-2 Clause

Function-name-2 can be defined as UPSI-0 through UPSI-7 or as SYSTEM-SHUTDOWN.

User Program Status Indicator (UPSI): Function-name-2 can define eight 1-byte program switches, UPSI-0 through UPSI-7.

Each UPSI is a User Program Status Indicator switch. At least one condition-name must be associated with each UPSI switch specified. UPSI-0 through UPSI-7 are COBOL names that identify program switches defined outside the COBOL program at object time. Their contents are considered to be alphanumeric. A value of zero is off; a value of one is on.

Each switch represents one byte from the 8-character SWS parameter of the control language CHGJOB, SBMJOB, JOB, and JOBD commands as follows:

```

UPSI-0 First byte (leftmost)
UPSI-1 Second byte
UPSI-2 Third byte
.
.
.
UPSI-7 Eighth byte (rightmost)

```

One condition-name must be associated with each function-name-2; a second condition-name is optional. One condition-name can be associated with the ON status; another can be associated with the OFF status. Establishing condition-names for the ON or OFF status of a switch permits testing the setting of that switch.

Each condition-name is formed according to the rules for a user-defined word, and the condition-name must contain at least one alphabetic character.

In the Procedure Division, the UPSI switch status is tested through the associated condition-name(s). Each condition-name is the equivalent of a level-88 item. The associated mnemonic-name, if specified, is considered the conditional variable and can be used for qualification.

Programming Notes: UPSI switches are useful for processing special conditions within a program, such as year-beginning or year-ending processing. At the beginning of the Procedure Division, an UPSI switch can be tested; if it is ON, the special branch is taken.

SYSTEM-SHUTDOWN: SYSTEM-SHUTDOWN is an internal switch that is set to ON status when the system operator causes the system to be in a shutdown-pending state or when the job is being canceled in a controlled manner. The associated ON or OFF condition-names can be referenced anywhere a condition-name is valid. Their status cannot be altered by the program.

Coding Example

This coding example assigns mnemonic-names to some commonly used function-names in the SPECIAL-NAMES paragraph.

```
SPECIAL-NAMES.  SYSTEM-CONSOLE IS SYSTM,  
                REQUESTOR IS WORK-STATION,  
                COI IS NEXT-PAGE,  
                LOCAL-DATA IS LOCAL-DATA-AREA,  
                ATTRIBUTE-DATA IS ATTRB-DATA,  
                SYSTEM-SHUTDOWN IS SHUTDOWN-SWITCH,  
                ON STATUS IS SHUTDOWN-PENDING,  
                UPSI-0 IS UPSI-SWITCH-0,  
                ON STATUS IS UO-ON,  
                OFF STATUS IS UO-OFF,  
                UPSI-1 IS UPSI-SWITCH-1,  
                ON STATUS IS U1-ON,  
                OFF STATUS IS U1-OFF,  
                IBM-ASCII IS STANDARD-1,  
                CURRENCY-SIGN IS "Y"
```

Alphabet-Name Clause

The alphabet-name clause provides a means of relating an alphabet-name to a specified character code set or collating sequence.

The alphabet-name specifies a collating sequence in one of the following:

- The PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph
- The COLLATING SEQUENCE phrase of the SORT or MERGE statement.

The EBCDIC collating sequence is used when NATIVE is specified or when the alphabet-name clause is omitted.

The ASCII (American National Standard Code for Information Interchange) collating sequence is used when STANDARD-1 is specified.

Literal Phrase: The literal phrase of the alphabet-name clause processes internal data in collating sequences other than NATIVE or STANDARD-1.

When the literal phrase is specified, the collating sequence to be used is specified by the user according to the following rules:

- The order in which literals appear specifies the ordinal number, in ascending sequence, of the character(s) in this collating sequence.
- Each numeric literal specified must be an unsigned integer and must have a value from 1 through 256 (the maximum number of characters in the EBCDIC character set). The value of each literal specifies the relative position of a character within the EBCDIC character set. For example, the literal 112 represents the EBCDIC character ?, the literal 234 represents the EBCDIC character Z, the literal 241 represents the EBCDIC numeric character 0.
- Each character in a nonnumeric literal represents that character in the EBCDIC set. If the nonnumeric literal contains more than one character, each character, starting with the leftmost, is assigned a successively ascending position within this collating sequence.
- Any EBCDIC characters not explicitly specified assume positions in this collating sequence higher than any of the explicitly specified characters. The relative order of the unspecified characters within the EBCDIC set remains unchanged.
- Within one alphabet-name clause, a given character must not be specified more than once.
- Each nonnumeric literal associated with a THROUGH or ALSO phrase must be one character in length.
- When the THROUGH phrase is specified, the contiguous EBCDIC characters beginning with the character specified by literal-1 and ending with the character specified by literal-2 are assigned successively ascending positions in this collating sequence. This sequence may be either ascending or descending within the original EBCDIC sequence. For example, if the characters Z through S are specified, then for this collating sequence the ascending values are: ZYXWVUTS
- When the ALSO phrase is specified, the EBCDIC characters specified as literal-1, literal-3, literal-4, and so on are assigned to the same position in this collating sequence. For example, if "D" ALSO "N" ALSO 112 ALSO "%" is specified, then for this collating sequence the characters D, N, ?, and % are all considered to be in the same position in the collating sequence.

If specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with hex 00 and hex FF respectively.

After all clauses in the SPECIAL-NAMES paragraph are processed, the character having the highest ordinal position in this collating sequence is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position because the ALSO phrase is specified, the last character specified is considered to be the HIGH-VALUE character for procedural statements such as DISPLAY, or as the sending field in a MOVE statement. If the ALSO phrase example given above were specified as the high-order characters of the collating sequence, then the HIGH-VALUE character would be %.

After all clauses in the SPECIAL-NAMES paragraph are processed, the character having the lowest ordinal position in this collating sequence is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position because the ALSO phrase is specified, the first character specified is the LOW-VALUE character. If the ALSO phrase example given above were specified as the low-order characters of the collating sequence, then the LOW-VALUE character would be D.

Alphabet-Name Clause Examples: The following examples illustrate some uses for the alphabet-name clause.

IF PROGRAM COLLATING SEQUENCE IS USER-SEQUENCE; if the alphabet-name clause is specified as USER-SEQUENCE IS "D", "E", "F"; and if two Data Division items are defined as follows:

```
01 ITEM-1 PIC X(3) VALUE "ABC".
01 ITEM-2 PIC X(3) VALUE "DEF".
```

then the comparison IF ITEM-1 > ITEM-2 is true.

Characters D, E, and F are in ordinal positions 1, 2, and 3 of this collating sequence. Characters A, B, and C are in ordinal positions 197, 198, and 199 of this collating sequence.

If the alphabet-name clause is USER-SEQUENCE IS 1 THRU 247, 251 THRU 256, "7", ALSO "8", ALSO "9"; if all 256 EBCDIC characters have been specified; and if the two Data Division items are specified as follows:

```
01 ITEM-1 PIC X(3) VALUE HIGH-VALUE.
01 ITEM-2 PIC X(3) VALUE "787".
```

then both of the following comparisons are true:

```
IF ITEM-1 = ITEM-2
IF ITEM-2 = HIGH-VALUE
```

They compare as true because the values "7", "8", and "9" all occupy the same position (HIGH-VALUE) in this USER-SEQUENCE collating sequence.

If the alphabet-name clause is specified as USER-SEQUENCE IS "E", "D", "F" and a table in the Data Division is defined as follows:

```
05 TABLE A OCCURS 6 ASCENDING KEY IS  
   KEY-A INDEXED BY INX-A.  
   10 FIELD-A .  
   10 KEY-A . .
```

and if the contents in ascending sequence of each occurrence of KEY-A are A, B, C, D, E, G, then the results of the execution of a SEARCH ALL statement for this table will be invalid because the contents of KEY-A are not in ascending order. The proper ascending order would be E, D, A, B, C, G.

CURRENCY SIGN Clause

The literal that appears in the CURRENCY SIGN clause defines the currency symbol to be used in the PICTURE clause. The literal must be a one-character nonnumeric literal and must not be any of the following characters:

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

When the CURRENCY SIGN clause is omitted, only the dollar sign (\$) may be used as the PICTURE symbol for the currency sign.

DECIMAL-POINT IS COMMA Clause

When the DECIMAL-POINT IS COMMA clause is specified, the functions of the period and the comma are exchanged in PICTURE character-strings and in numeric literals.

Input-Output Section

The Input-Output Section defines each file, identifies its external storage medium, assigns the file to one or more input/output devices, and also specifies information needed for efficient transmission of data between the external medium and the COBOL program.

Files

System/38 has two categories of files: data base files and device files.

Data Base Files

Data base files allow information to be permanently stored on the system. Multiple programs can access this information in different ways.

A data base file is subdivided into groups of records called members. Every file has at least one member.

There are two types of data base files: physical files and logical files.

Physical Files: A physical file is a file that actually contains data records. This makes physical files similar to disk files on other systems. A physical file can contain only fixed-length records, all of which have the same format.

Logical Files: A logical file is a data base file through which data from one or more physical files can be accessed. The format and organization of this data is different from that of the data in the physical file(s). Each logical file can define a different access path (index) for the data in the physical file(s). Each logical file can exclude and reorder the fields defined in the physical file(s).

Device Files

A device file reads from or writes to a device attached to the system. A device file controls the transfer of data between the physical device and the program.

This manual uses the term *file* as a device file or a member in a data base file.

DDM Files

Distributed Data Management (DDM) allows you to access data files that reside on remote IBM System/36 and System/38 systems. DDM files are supported by the COBOL compiler. You can retrieve, add, update or delete data records in a file that resides on another system. In addition, a remote system can access your System/38 data base for record retrieval.

For more information about accessing remote files, refer to the *IBM System/38 Distributed Data Management User's Guide*.

Paragraphs

The Input-Output Section is divided into two paragraphs: the FILE-CONTROL paragraph, which names and associates the files with the external media, and the I-O-CONTROL paragraph, which defines special input/output techniques to be used.

Format

```
[ INPUT-OUTPUT SECTION.
  FILE-CONTROL. {file-control-entry} . . .
  [ I-O-CONTROL. input-output-control-entry ] ] .
```


The exact contents of the Input-Output Section depend on the file organization and access methods used to process the file. The following summary gives some background for the file processing techniques available in System/38 COBOL.



File Processing Summary

The method used to process a file in a COBOL program depends on the data organization of the file and on the access mode used.

Appendix J summarizes which clauses and statements are required and which clauses and statements are optional for each access mode and device.

The following paragraphs describe both the types of data organization, and the access modes available in COBOL. See Chapter 9 for information about COBOL file processing in relation to System/38 file processing.

Data Organization

In a COBOL program, data organization can be sequential, indexed, relative, or TRANSACTION.

Records can be fixed or variable in length. For all files other than tape, variable length records are stored as fixed length records of the maximum size specified for the file.

Sequential Organization: With this organization, records are placed in the file consecutively, without keys, in the order they are written (arrival sequence). Once established, this relationship does not change, with the exception that a file can be extended. Both data base files and device files can have sequential organization.



Indexed Organization: With this organization, each record in the file has one embedded key that is associated with an index. The index provides a logical path to the data records according to the contents of the associated embedded record key data item (key sequence).

When records are inserted, updated, or deleted, they are identified solely by the value of their record key. Thus, the value in each record key data item must be unique and must not be changed when the record is updated. The key used for any specific input/output request is known as the *key of reference*.

Only data base files can have indexed organization.



IBM Extension

A logical file that is opened for OUTPUT does not remove all records in the physical file on which it is based. Instead, the file is opened to allow only write operations, and the records are added to the file.

End of IBM Extension

Relative Organization: With this organization, each record in the file is identified by its relative record number. The file can be thought of as a serial string of areas, each of which can contain one record. Each of these areas is identified by a relative record number; record storage and retrieval are based on this number. For example, the first record area is addressed by relative record number 1, and the tenth is addressed by relative record number 10, whether or not records have been written in the second through ninth record areas. Relative files must be assigned to DISK or DATABASE.

New relative files opened for OUTPUT are initialized with all records deleted. In the absence of command language override, the number of records in a newly created file is the number of records specified at file creation time including all increments. Any attempt to extend a relative file beyond its current size results in a boundary violation.

Relative record number processing can be used for a physical file or for a logical file that is based on only one physical file.

IBM Extension

TRANSACTION Organization: Work station and data communication files can have TRANSACTION organization. See Chapter 7 for a discussion of this organization.

End of IBM Extension

Access Modes

Access mode is a COBOL term that defines the manner in which data in a logical or physical file is to be processed. The three access modes are sequential, random, and dynamic.

Sequential Access Mode: This access method allows records of a file to be read and written in a serial manner. The order of reference is implicitly determined by the position of a record in the file.

Random Access Mode: This access method allows records to be read and written in a user-specified manner. The control of successive references to the file is expressed by specifically defined keys supplied by the user.

Dynamic Access Mode: This access method allows a specific input/output request to determine the access mode. Thus records can be processed sequentially and/or randomly.

Access Mode Allowed for Each File Type

Sequential Files: Files with sequential organization can be accessed only sequentially. The sequence in which records are accessed is the order in which the records were originally written (arrival sequence).

Indexed Files: All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is determined by the RECORD KEY value.

In the random access mode, the sequence in which records are accessed is controlled by the user. The desired record is accessed by placing the value of its record key in the RECORD KEY data item defined for that file.

In the dynamic access mode, the user can change from sequential access to random access by using appropriate input/output statements.

Relative Files: All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is the ascending order 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 user. The desired record is accessed by placing its relative record number in a RELATIVE KEY data item.

In the dynamic access mode, the user can change from sequential access to random access by using appropriate input/output statements.

TRANSACTION Files: See Chapter 7 for a discussion of access mode considerations for TRANSACTION files.

FILE-CONTROL Paragraph

The FILE-CONTROL paragraph contains one or more file-control entries. A file-control entry associates a file in the COBOL program with an external medium, and this entry allows specification of file organization, access mode, and other information. The format of a file-control entry varies with the type of file described. The formats for the FILE-CONTROL paragraph are as follows:

Format 1 – Sequential File Entries (READER, PUNCH, PUNCHPRINT, PRINT, PRINTER, TAPEFILE, DISKETTE, FORMATFILE, DISK, DATABASE)

```

SELECT [ OPTIONAL ] file-name
      ASSIGN TO assignment-name-1 [ , assignment-name-2 ] ...
      [ RESERVE integer-1 [ AREA
                          AREAS ] ]
      [ ORGANIZATION IS SEQUENTIAL ]
      [ ACCESS MODE IS SEQUENTIAL ]
      [ FILE STATUS IS data-name-1 ] .
  
```

Format 2 – Indexed File Entries (DISK, DATABASE)

```

SELECT file-name
      ASSIGN TO assignment-name-1 [ , assignment-name-2 ] ...
      [ RESERVE integer-1 [ AREA
                          AREAS ] ]
      ORGANIZATION IS INDEXED
      [ ACCESS MODE IS { SEQUENTIAL
                        RANDOM
                        DYNAMIC } ]
      RECORD KEY IS { EXTERNALLY-DESCRIBED-KEY } [ WITH DUPLICATES ]
                    data-name-2
      [ FILE STATUS IS data-name-1 ] .
  
```

Format 3 – Relative File Entries (DISK, DATABASE)

SELECT file-name

ASSIGN TO assignment-name-1 [, assignment-name-2] ...]

[RESERVE integer-1 [AREA AREAS]]

ORGANIZATION IS RELATIVE

[ACCESS MODE IS { SEQUENTIAL [, RELATIVE KEY IS data-name-3] }
{ RANDOM
DYNAMIC } , RELATIVE KEY IS data-name-3]

[FILE STATUS IS data-name-1] .

Format 4 – Sort or Merge File Entries

SELECT file-name [ASSIGN TO assignment-name-1 [, assignment-name-2] ...] .

IBM Extension

Format 5 - TRANSACTION File Entries (WORKSTATION)

See Chapter 7 for a discussion of this format.

End of IBM Extension

FILE-CONTROL Paragraph – General Considerations

Each file described in an FD or SD entry in the Data Division must be described in only one entry in the FILE-CONTROL paragraph. Each file specified in a file-control entry must have a file description in the Data Division.

The keyword FILE-CONTROL can appear only once, at the beginning of the FILE-CONTROL paragraph. The word FILE-CONTROL must begin in Area A, and it must be followed by a period and a space.

Each file-control entry must begin in Area B with a SELECT clause. The order in which other clauses appear is not significant.

Each clause within a file-control entry can optionally be separated from the next by a comma or semicolon followed by a space. Each file-control entry ends with a period and a space.

Each data-name must appear in a Data Division data description entry. Each data-name can be qualified but cannot be subscripted or indexed.

SELECT Clause

Each file-name specified in a SELECT clause must have an FD or SD entry in the Data Division. A file-name must conform to the rules for a COBOL user-defined name, must contain at least one alphabetic character, and must be unique within this program.

Sequential File Considerations: The OPTIONAL phrase can be specified only for input files with sequential organization. It must be specified for input files that are not necessarily present each time the program is executed.

ASSIGN Clause

The ASSIGN clause associates a file with an external medium. The assignment-name makes the association between the file and the external medium. For sort or merge files (associated with an SD entry), no external medium is used. The related ASSIGN clause is only validity checked. It is not actually used for I-O.

Assignment-name consists of 3 parts:

- Device
- System/38 file name
- Attribute.

It has the following general structure:

Device [- System/38 file name [- attribute]]

Device: This part of assignment-name specifies the type of device that the file will use. The compiler can then check whether the file is described and used in a consistent manner. See “Device Independence/Device Dependence” in Chapter 9 for further information.

The compiler does not check whether the device associated with the external file is of the type specified in the device portion of assignment-name. For example, assignment-name could be TAPEFILE-ABCD and ABCD could be created with a Create Card File (CRTCRDF) CL command. The compiler would provide no diagnostics unless the I-O verbs were used in an inconsistent manner for TAPEFILE. At execution time, CPF (control program facility) could either issue an escape message or ignore the function if it was not applicable to the device. See the *CPF Programmer's Guide* for further information on overriding files.

IBM Extension

The device type can be changed at execution time with the DEV parameter of the OVRxxxxF CL command. To ensure consistent results, the device type associated with the DEV parameter should be the same as that specified for assignment-name.

End of IBM Extension

Device can be any of the following:

Device	Associated File
READER	Card file
PUNCH	Card file
PUNCHPRINT	Card file
PRINT	Card file
PRINTER ¹	Printer file
FORMATFILE ²	Printer file
TAPEFILE	Tape file
DISKETTE	Diskette file
DISK ³	Any physical data base file or single format logical data base file
DATABASE ⁴	Any data base file
WORKSTATION	Display file, communications file, binary synchronous communications file, or mixed file.

For more information on how to use externally described printer files see "FORMATFILE Files" in Chapter 9.

Note: See "DISK and DATABASE File Considerations" in Chapter 9 for further information.

System/38 File Name: This part of assignment-name must be an unhyphenated, 1- through 10-character system name of the actual external file (physical or logical data base, or device). This external file has to be created before compiling the program only when it is used by a COPY statement, DDS (data description specifications) or DD format, within this program.

For data base files, the member name cannot be specified in the program. If a member other than the first member is to be specified, the Override with

-
- 1 PRINTER should be specified for program described printer files only.
 - 2 FORMATFILE should be specified for externally described printer files only.
 - 3 When DISK is the device, data base extensions cannot be used.
 - 4 When DATABASE is the device, externally described data and data base extensions can be used.

Data Base File (OVRDBF) CL command must be used at execution time to specify the member name.

This System/38 file name is the name of the CPF object that is displayed by the Display Program References (DSPPGMREF) command. Since no external medium is used for an SD file, the DSPPGMREF command does not list any files defined for an SD file.

The System/38 file name can be changed at execution time with the TOFILE parameter of the OVRxxx F CL command. To ensure consistent results, the device type associated with the TOFILE parameter should be the same as that specified for assignment-name.

Attribute: This part of assignment-name can be one of the following:

- – hopper [– association]
- – SI.

Hopper must be either P or S to specify the primary or secondary hopper for card device files. If neither P nor S is specified for a card device file, the HOPPER parameter on the Create Card File (CRTCRDF) or Change Card File (CHGCRDF) CL commands is used.

Association must be any single-digit number from 0 through 9. It can be used only if the primary (P) hopper is specified for the file. All unit record card files that have the same association number are assigned to the same unit record card device, and must use the same external file name (see Appendix D).

SI indicates that a separate indicator area has been specified in the DDS for a FORMATFILE or WORKSTATION file. See “Indicators” in Chapter 7 for more information on the use of the SI attribute.

See “Device Independence/Device Dependence” in Chapter 9 for further information on the ASSIGN clause.

The valid entries for each field of the assignment-name vary with the device. The valid combinations of fields are shown in Figure 3-2.

In formats 1, 2, and 3, the second and subsequent assignment-names are syntax-checked, but are treated as documentation. In format 4, the entire ASSIGN clause is syntax-checked, but is treated as documentation.

RESERVE Clause

The RESERVE clause is syntax-checked, but is treated as documentation.

ORGANIZATION Clause

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

IBM Extension

For data base files, the ORGANIZATION clause indicates the current program usage of the file in the program. Therefore, the same data base file can use SEQUENTIAL, INDEXED (assuming a keyed sequence access path exists), or RELATIVE in the ORGANIZATION clause. This is true regardless of what is specified in other programs that use this file.

End of IBM Extension

Note: A keyed sequence access path is always created when a key is specified in the DDS that was used as input to the Create Physical File (CRTPF) or the Create Logical File (CRTLF) CL command.

Device	System/38 File Name	Default System/38 File Name	Hopper	Association	SI
READER	0	QCARD96	0	0	N
PUNCH	0	QCARD96	0	0	N
PUNCHPRINT	0	QCARD96	0	0	N
PRINT	0	QCARD96	0	0	N
PRINTER	0	QPRINT	N	N	N
FORMATFILE	R		N	N	O
TAPEFILE	0	QTAPE	N	N	N
DISKETTE	0	QDKT	N	N	N
DISK	R		N	N	N
DATABASE	R		N	N	N
WORKSTATION	R		N	N	O

R = Required
 O = Optional
 N = Not Allowed

Figure 3-2. Valid Entries for the Assignment-Name

Sequential File Considerations: When ORGANIZATION IS SEQUENTIAL is specified or implied, a predecessor-successor relationship of the records in the files is established by the order in which records are placed in the file when it is created or extended (arrival sequence access path).

Indexed File Considerations: When ORGANIZATION IS INDEXED is specified, the position of each logical record in the file is determined by the key sequence access path created with the file and maintained by the system. The access path is based on an embedded key within the file's records.

Relative File Considerations: When ORGANIZATION IS RELATIVE is specified, the position of each record in the file is determined by its relative record number within the arrival sequence access path.

ACCESS MODE Clause

The ACCESS MODE clause defines the manner in which the records of the file are made available for processing. When this clause is omitted, ACCESS IS SEQUENTIAL is assumed.

Sequential File Considerations: For files with sequential organization, records in the file are accessed in the order they are written when the file is created or extended (arrival sequence). Whether ACCESS IS SEQUENTIAL is specified or omitted, sequential access is always assumed.

Indexed File Considerations: For files with indexed organization, the access mode can be SEQUENTIAL, RANDOM, or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, records in the file are accessed in the sequence of ascending record key values within the index.

IBM Extension

When using an externally described file, if the DDS keyword DESCEND is used when the field is specified as a key field, the records in the file are accessed in the sequence of descending record key values within the index. Either the DESCEND keyword, or the ASCEND keyword (if DESCEND is not specified) appears under the heading RETRIEVAL in a comment table in the COBOL source program listing.

End of IBM Extension

When ACCESS IS RANDOM is specified, the value placed in the RECORD KEY data item specifies the record to be accessed.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request.

Relative File Considerations: For files with relative organization, the access mode can be SEQUENTIAL, RANDOM, or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, records in the file are accessed in the ascending sequence of relative record numbers in the arrival sequence access path.

When ACCESS IS RANDOM is specified, the value placed in the RELATIVE KEY data item specifies the record to be accessed.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request.

RELATIVE KEY Phrase: The RELATIVE KEY phrase specifies the relative record number for a specific record in a relative file.

Data-name-3 is the RELATIVE KEY data item. It must be defined as an unsigned integer data item and must not be defined in a record description entry associated with this relative file. That is, the RELATIVE KEY is not part of the record.

When ACCESS IS SEQUENTIAL is specified, the RELATIVE KEY phrase need not be specified unless the START statement is used. When the START statement is used, the system uses the contents of the RELATIVE KEY data item to determine the record at which sequential processing is to begin.

If a value is placed in the RELATIVE KEY data item and a START statement is not used, the value is ignored and processing begins with the first record in the file.

IBM Extension

When the file is opened, the POSITION parameter on the OVRDBF CL command can be used to set the current record pointer. This causes processing to begin with a record other than the first record. See the *CL Reference Manual* for further information.

End of IBM Extension

When ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, the RELATIVE KEY phrase must be specified. For each random processing request, the contents of the RELATIVE KEY data item are used to communicate a relative record number to the system.

TRANSACTION File Considerations: See Chapter 7.

RECORD KEY Clause (Indexed File)

The RECORD KEY clause must be specified for an indexed file. The RECORD KEY clause specifies the data item within the record that is the record key for an indexed file. The values contained in the record key data item must be unique among records in the file.

IBM Extension

The DUPLICATES phrase can only be specified for files assigned to DATABASE. This allows the file to have keys with the same values. If the file has multiple formats, two keys in different formats have the same values only when the key lengths and the contents of the keys are the same.

For example, given a file with the following two formats:

Format F1 with keys A, B, C
Format F2 with keys A, B, D.

If fields C and D are the same length, have the same data type, and have the same values, the file would contain two records with a duplicate key. The term *duplicate key* applies only to a complete record key for the format. A record key for the format consists of the key field(s) defined for a DDS format for records residing on the data base. The term does not apply to the common key for the file (only fields A and B in the above example).

Users can indicate DUPLICATES on the RECORD KEY clause. A file status of 95 is returned after a successful open when:

- The DUPLICATES phrase is specified in the COBOL program and the file was created with UNIQUE specified in DDS.
- The DUPLICATES phrase is not specified in the COBOL program and the file was created allowing nonunique keys.

Processing files when either of these conditions exist can cause unpredictable results.

To ensure that the proper duplicate record is updated or deleted in a file that allows duplicates and is processed randomly, the last input/output statement executed prior to the execution of the REWRITE or DELETE statement must be a successfully executed READ statement for the record to be deleted or rewritten.

If the DDS file level keyword LIFO (last-in-first-out) is specified, the duplicate records within a physical file are retrieved in a last-in-first-out order.

End of IBM Extension

Data-name-2 is the RECORD KEY data item. It must be described as a fixed-length alphanumeric item within a record description entry associated with the file. The length of the record key is restricted; the key length, in characters, plus the number of fields cannot exceed 120. See the *CPF Reference Manual – DDS* for more information.

IBM Extension

The RECORD KEY data item, data-name-2, can be a numeric item when the file is assigned to a DATABASE device type. The numeric item can have a usage of DISPLAY, COMP (COMP-3), or COMP-4.

Depending on the keywords specified for the data item in DDS, the keyed sequence access path can be by algebraic value. See the ABSVAL, DIGIT, SIGNED, and ZONE keywords in the *CPF Reference Manual – DDS*. If one of these keywords is specified, its name appears in a comment table in the COBOL source listing under the heading TYPE. If no keyword is specified, the table entry is the data type specified in DDS. The table entry AN indicates that the data type is alphanumeric (specified in DDS as A). The

table entry N indicates that the data type is numeric (specified in DDS as P, S, or B).

The keywords specified for the data item in DDS can modify record sequence. See the ALTSEQ, DIGIT, and ZONE keywords in the *CPF Reference Manual - DDS*. If none of these keywords are specified, the records are ordered according to the EBCDIC collating sequence.

End of IBM Extension

The data description of data-name-2 and its relative location within the record must be the same as the ones used when the file was defined in DDS.

The record description that defines data-name-2 will always be used to access the record key field for the I-O operation.

IBM Extension

The reserved word EXTERNALLY-DESCRIBED-KEY can specify that the key(s) for this file are those that are externally described in DDS. The keys are determined by the record formats that are copied by the COPY statement, DDS or DD format, under the FD for this file.

The key can start at different offsets within the buffer for each format. In this situation, care must be used when changing from one record format to another, using a random READ or START statement. The key must be placed in the record format at the correct offset in the format that will be used in the random access of the file. Unpredictable results can occur if the key for the desired record is based on data that was part of the last record read. This is because the movement of the data to the key field can involve overlapping fields.

The key within a format can be made up of multiple, noncontiguous (not adjacent) fields. When using EXTERNALLY-DESCRIBED-KEY for a logical file, the key fields defined for a record format in DDS must also be fields defined in that format. Therefore, fields renamed in DDS, or fields that are part of concatenated fields in DDS cannot be used as keys. Only those record formats copied in within the FD for the file should be referenced by the FORMAT phrase. If a format is referenced that is defined within the file, but that format has not been copied into the program, the key is built using the key field(s) defined for the first record format that was copied. This can cause unpredictable results.

End of IBM Extension

FILE STATUS Clause

The FILE STATUS clause allows the user to monitor the execution of each input/output request for the file.

Data-name-1 is the status key data item. Data-name-1 must be defined in the Data Division as a two-character alphanumeric item and must not be defined in the File Section.

When the FILE STATUS clause is specified, the system moves a value into the status key data item after each input/output request that explicitly or implicitly refers to this file. The value indicates the execution status of the statement. When the compiler generates code to block output records or unblock input records, file status values that are caused by CPF exceptions are set only when a block is processed. See Appendix I for a description of the possible values. See Chapter 9 for more information on blocking output records and unblocking input records.

IBM Extension

An extended file status data item may be specified for TRANSACTION file processing. See Chapter 7 for more information.

End of IBM Extension

I-O-CONTROL Paragraph

The I-O-CONTROL paragraph specifies when checkpoints are to be taken and what storage areas are to be shared by different files and optimization techniques. The I-O-CONTROL paragraph is optional in a COBOL program.

Format

```
[ I-O-CONTROL.  
  [ RERUN ON assignment-name  
    EVERY integer-1 RECORDS OF file-name-1 ] . . .  
  [ SAME [ RECORD  
            SORT  
            SORT-MERGE ] AREA FOR file-name-2 { , file-name-3 } . . . ] . . .  
  [ MULTIPLE FILE TAPE CONTAINS  
    file-name-4 [ POSITION integer-2 ]  
    [ file-name-5 [ POSITION integer-3 ] ] . . . ] . . . .  
  
  [ COMMITMENT CONTROL FOR  
    file-name-6  
    [ , file-name-7 ] . . . ] . ]
```

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

Each clause within the `I-O-CONTROL` entry can optionally be separated from the next by a comma or semicolon followed by a space. The clauses, when present, must be specified in the order shown. Clauses can be specified on the same line as the `I-O-CONTROL` paragraph header, or on separate lines. The `I-O-CONTROL` entry ends with a period followed by a space.

RERUN Clause

The `RERUN` clause is syntax-checked, but is treated as documentation.

Assignment-Name: This name can be any user-defined word.

SAME Clause

The SAME clause specifies that two or more files are to use the same main storage area during processing. The files named in a SAME clause need not have the same organization or access.

The following discussion describes only the SAME RECORD AREA and SAME AREA clauses. The SAME SORT AREA and SAME SORT-MERGE AREA clauses are discussed under "SORT/MERGE" in Chapter 6.

The SAME RECORD AREA clause and SAME AREA clause are intended to make efficient use of main storage. However, the virtual storage architecture of System/38 eliminates the need for these clauses, and the clauses are supported for compatibility rather than for performance. Use of the SAME RECORD AREA actually degrades performance.

The SAME RECORD AREA clause specifies that two or more files are to use the same main storage area for processing the current record. All the files can be open at the same time. A record in the shared storage area is considered to be both a record of each opened output file in this SAME RECORD AREA clause, and a logical record of the most recently read input file in this SAME RECORD AREA clause.

More than one SAME RECORD AREA clause can be included in a program; however, the following restriction applies:

- A specific file-name must not appear in more than one SAME RECORD AREA clause.

The SAME AREA clause is syntax-checked, but is treated as documentation. However, the following restrictions apply:

- A specific file-name must not appear in more than one SAME AREA clause.
- If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, that SAME RECORD AREA clause can contain additional file-names that do not appear in that SAME AREA clause.
- Only one of the files for which the SAME AREA clause is specified can be open at one time. This rule takes precedence over the SAME RECORD AREA rule that all the files can be open at the same time.

Programming Notes: The SAME RECORD AREA clause allows transfer of data from one file to another with no explicit data manipulation because the input/output record areas of named files are identical, and all are available to the user.

MULTIPLE FILE TAPE Clause

The MULTIPLE FILE TAPE clause is syntax-checked, but is treated as documentation. This clause specifies that two or more files share the same reel of tape. The function is provided by the system through the use of command language. See CRTTAPF, CHGTAPF, and OVRTAPF commands in the *CL Reference Manual*.

COMMITMENT CONTROL Clause

The COMMITMENT CONTROL clause specifies the files that will be placed under commitment control when they are opened. These files will then be affected by the COMMIT and ROLLBACK statements. The COMMIT statement allows the synchronization of changes to data base records while preventing other jobs from modifying those records until the COMMIT is complete. The ROLLBACK statement provides a method of cancelling changes made to data base files when those changes should not be made permanent.

The COMMITMENT CONTROL clause can specify only files assigned to a device type of DATABASE. Files under commitment control may have an organization of sequential, relative or indexed, and may have any access mode valid for a particular organization.

The system locks records contained in files under commitment control when these records are accessed. Records remain locked until released by a COMMIT or ROLLBACK statement. For more information about record locking for files under commitment control, see "Commitment Control Considerations" in Chapter 9.

Programming Note: Always try to use files in a consistent manner to avoid record locking problems, and to avoid reading records that have not yet been permanently committed to the data base. Typically, a file should either always be accessed under commitment control or never be accessed under commitment control.

Chapter 4. Data Division

Data Division Concepts

The Data Division of a COBOL source program describes all the data to be processed by the object program. Two types of data can be processed: external data and internal data.

External Data

External data is contained in files. A file is a collection of data records existing on some input/output device. A file can be thought of as a group of physical records; it can also be thought of as a group of logical records. The Data Division source statements describe the relationship between physical and logical records. (See the Glossary for definitions of these items.)

A physical record is a unit of data that is treated as an entity when it is moved into or out of auxiliary storage. The size of a physical record is determined by the particular input/output device on which it is stored. The size does not necessarily have a direct relationship to the size or content of the logical information contained in the file.

A logical record is a unit of data whose subdivisions have a logical relationship. A logical record can itself be a physical record (that is, be contained completely in one physical unit of data), or several logical records can be contained within one physical record.

Record description entries, which follow the FD (file description) entry for a specific file, describe the logical records in the file. These entries also describe the category and format of data within each field of the logical record and different values the data might be assigned.

The FD entry specifies the physical aspects of the data such as the size relationship between physical and logical records, the size and name(s) of the logical record(s), and labeling information.

Once the relationship between physical and logical records has been established, only logical records are made available to the COBOL program. Thus, in this manual, a reference to records means logical records unless the term physical records is used.

Internal Data

Program logic can develop additional data within storage. Such data is called internal data.

The concept of logical records applies to internal data as well as to external data. Internal data can thus be grouped into logical records and be defined by a series of record description entries. Items that need not be so grouped can be defined in independent data entries.

Data Relationships

The relationships of all data to be used in a program are defined in the Data Division through a system of level indicators and level-numbers.

A level indicator, together with its descriptive entry, identifies each file description in a program. Level indicators are the highest level of any data hierarchy with which they are associated.

A level-number, together with its descriptive entry, indicates the properties of specific data. Level-numbers can be used to describe a data hierarchy. They can indicate that this data has a special purpose, and while they can be associated with and be subordinate to level indicators, they can also be used independently to describe internal data or data common to two or more programs.

Data Division Organization

The Data Division is divided into three sections: the File Section, the Working-Storage Section, and the Linkage Section. Each section has a specific logical function within a COBOL source program, and each can be omitted from the source program when that logical function is not needed.

Format

DATA DIVISION.

[FILE SECTION.

[file-description-entry, { record-description-entry } . . .] . . .

[sort-merge-file-description-entry, { record-description-entry } . . .] . . .]

[WORKING-STORAGE SECTION.

[data-description-entry] . . .

[record-description-entry] . . .]

[LINKAGE SECTION.

[data-description-entry] . . .

[record-description-entry] . . .]

The Data Division must begin with the words DATA DIVISION followed by a period and a space.

In the source program, the Data Division sections must appear in the order shown.

Coding Example

SEQUENCE		CONT		A	B
(PAGE)	SERIAL	1	2	3	4
003	010	DATA	DIVISION.		
	020	FILE	SECTION.		
	030	FD	FILE-NAME		
	040		BLOCK	→	
	050		RECORD	→	
	060		LABEL RECORD	→	
	070		LINKAGE	→	
	080		DATA RECORD IS	→	
	090	01	DESCRIPTION.		
	100	WORKING-STORAGE	SECTION.		
	110	01	NAME-DESCRIPTION.		
	120	01	RECORD-DESCRIPTION.		
	13				

Sample Data Division Entries

```

1 ..... 2 ..... 3 ..... 4 ..... 5 ..... 6 ..... 7
DATA DIVISION.
FILE SECTION.
FD INPUT-DATA
BLOCK CONTAINS 1 RECORDS
RECORD CONTAINS 80 CHARACTERS
LABEL RECORDS ARE STANDARD
DATA RECORDS ARE GEN-INFO SALES-DATA.
01 GEN-INFO.
03 EMPLOYEE-NAME.
05 FIRST-NAME PIC X(12).
05 LAST-NAME PIC X(12).
03 SOC-SEC-NUMBER PIC 9(9)
03 CHECK-SSN
REDEFINES SOC-SEC-NUMBER PIC X(9).
03 AGE PIC 99.
03 BIRTH-DATE.
05 B-MONTH PIC 99.
05 B-DAY PIC 99.
05 B-YEAR PIC 99.
03 ANNUAL-SALARY PIC 9(5)V99.
03 CHECK-SALARY
REDEFINES ANNUAL-SALARY PIC X(7)
* THIS REDEFINES WILL BE USED TO SEE IF THE FIELD IS BLANK.
03 RECORD-ID PIC X.
03 FILLER PIC X(31).
01 SALES-DATA.
03 SALES-SSN PIC 9(9).
03 SALES-LOCATION PIC XX.
88 MICHIGAN VALUE IS "MI"
88 EASTERN-REGION VALUES ARE "PA" "NY"
88 HEADQUARTERS VALUES ARE "BA" THRU "BZ".
03 TOTAL-COMMISSION PIC 9(5)V99.
03 RECORD-CODE PIC X.
03 FILLER PIC X(61).
FD REPORT-OUT
LABEL RECORDS ARE OMITTED
RECORD CONTAINS 132 CHARACTERS
LINAGE IS 66 LINES
FOOTING 6 LINES AT TOP 4 LINES AT BOTTOM 4
DATA RECORD IS PRINT-OUT.
01 PRINT-OUT PIC X(132).
WORKING-STORAGE SECTION.
01 RECORDS-IN PIC 9(6) VALUE ZEROS.
01 DECLARATIVE-ERRORS PIC 9(4) VALUE ZEROS
01 EOF-SW PIC X VALUE ZERO.
01 BAD-DATA-COUNTER PIC 9(3) VALUE ZEROS.
01 CHECK-IT PIC XX.
01 PRINT-FIELDS-EDITED.
03 FILLER PIC X(14) VALUE SPACES.
03 TOTAL-SALARY PIC $$$,$$$,99BB.
03 COMMISSION-COSTS PIC $$$,$$$,99B.
03 FILLER PIC X(65) VALUE ALL "-".
03 FILLER PIC X(12)
VALUE "----END---JOB"
01 SALARY-COUNTER PIC 9(6)V99 VALUE ZEROS.
01 COMMISSION-COUNTER PIC 9(6)V99 VALUE ZEROS

```

File Section

The File Section contains a description of all externally stored data (FD) and a description of each sort-merge file (SD) used in the program.

The File Section must begin with the header `FILE SECTION` followed by a period. The File Section contains file description entries and sort-merge file description entries. Each entry is followed by its associated record description entry (or entries).

In a COBOL program, the file description entries (beginning with the level indicators `FD` and `SD`) represent the highest level of organization in the File Section. The file description entry provides information about the physical structure and identification of a file, and gives the record-name(s) associated with that file. For further description of the format and the clauses required in a file description entry, see “File Description Entry” in this chapter. See “Data Division – SORT/MERGE” in Chapter 6 for a complete discussion of the sort-merge file description entry.

The record description entry consists of a set of data description entries that describe the records contained within a particular file. More than one record description entry can be specified; each is an alternative description of the same storage area. For the format and the clauses required within the record description entry, see “Data Description” in this chapter.

IBM Extension

The record description entry for a file can be specified using the `COPY` statement, `DDS` or `DD` format. This allows the field descriptions for a record format to be exactly as defined in `DDS`. Also, programs are easier to write because the record format description is maintained in only one place. See “SOURCE PROGRAM LIBRARY” in Chapter 6 for further information on the `COPY` statement, `DDS` or `DD` format.

End of IBM Extension

Data areas described in the File Section should not be considered available for processing unless the file containing the data area is open.

Working-Storage Section

The Working-Storage Section can contain description records that are not part of data files but are developed and processed internally. These records are used for report description, counters, and other functions necessary in processing data.

The Working-Storage Section must begin with the section header `WORKING-STORAGE SECTION` followed by a period. The Working-Storage Section contains record description entries and data description entries for noncontiguous data items.

Data elements in the Working-Storage Section that bear a definite hierarchical relationship to one another must be grouped into records structured by level-number.

Noncontiguous items in this section that bear no hierarchical relationship to one another need not be grouped into records provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each is defined in a separate data description entry that begins with the special level-number 77 or level-number 01. The format of the data description entry is the same as the format for the record description entry.

Linkage Section

The Linkage Section describes data made available from another program.

Record description entries and data description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved because the data area exists elsewhere. Any data description clause can be used to describe items in the Linkage Section with one exception: the VALUE clause cannot be specified for any items other than level-88 items. See "INTER-PROGRAM COMMUNICATION FEATURE" in Chapter 6 for additional information.

File Description Entry

In a COBOL program, the file description entry (FD entry) or the sort-merge file description entry (SD entry) is the highest level of organization in the File Section. Up to 99 FD and SD entries can be defined in a COBOL program.

Format 1 – Files (FORMATFILE, DATABASE, DISK, READER, PUNCH, PUNCHPRINT, PRINT)

```

[ FD file-name
  [ BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS
    CHARACTERS } ]
  [ RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS ]
  [ LABEL { RECORD IS
    RECORDS ARE } { STANDARD
    OMITTED }
    [ VALUE OF user-name-1 IS { data-name-1
      literal-1 }
      [ , user-name-2 IS { data-name-2
        literal-2 } ] . . . ]
  [ DATA { RECORD IS
    RECORDS ARE } data-name-3 [ , data-name-4 ] . . . ]
  { record-description-entry } . . . ] . . .

```

Format 2 – Files (DISKETTE)

```

[ FD file-name
  [ BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS
    CHARACTERS } ]
  [ RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS ]
  [ LABEL { RECORD IS
    RECORDS ARE } { STANDARD
    OMITTED }
    [ VALUE OF user-name-1 IS { data-name-1
      literal-1 }
      [ , user-name-2 IS { data-name-2
        literal-2 } ] . . . ]
  [ DATA { RECORD IS
    RECORDS ARE } data-name-3 [ , data-name-4 ] . . . ]
  [ CODE-SET IS alphabet-name ] .
  { record-description-entry } . . . ] . . .

```


Format 3 – Files (TAPEFILE)

[FD file-name
 [BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS
 { CHARACTERS } }]
 [RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]
 LABEL { RECORD IS
 { RECORDS ARE } } { STANDARD
 { OMITTED } }
 [VALUE OF user-name-1 IS { data-name-1 }
 { literal-1 }]
 [, user-name-2 IS { data-name-2 }] . . .]]
 [DATA { RECORD IS
 { RECORDS ARE } } data-name-3 [, data-name-4] . . .]
 [CODE-SET IS alphabet-name] .
 { record-description-entry } . . .] . . .]

Format 4 – Files (PRINTER)

```
[ FD file-name  
  [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS  
    { CHARACTERS } ]  
  [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]  
  [ LABEL { RECORD IS  
    { RECORDS ARE } { STANDARD  
    { OMITTED }  
    [ VALUE OF user-name-1 IS { data-name-1 }  
      { literal-1 }  
      [ , user-name-2 IS { data-name-2 }  
        { literal-2 } ] . . . ]  
  [ DATA { RECORD IS  
    { RECORDS ARE } data-name-3 [ , data-name-4 ] . . . ]  
  [ LINAGE IS { data-name-5 }  
    { integer-5 } LINES [ , WITH FOOTING AT { data-name-6 }  
    { integer-6 } ]  
    [ , LINES AT TOP { data-name-7 }  
    { integer-7 } ] [ , LINES AT BOTTOM { data-name-8 }  
    { integer-8 } ] ]  
  { record-description-entry } . . . ] . . . ]
```

Format 5 – Sort or Merge File Description

See “Data Division – SORT/MERGE” in Chapter 6 for the format of the sort-merge file description (SD entry).

IBM Extension

Format 6 – TRANSACTION File

See Chapter 7 for a discussion of this format.

End of IBM Extension

The file description entry must begin with the level indicator FD followed by a space.

The clauses that follow file-name are optional in many cases; the order of their appearance is not significant.

However, at least one record description entry must follow the FD entry. When more than one record description entry is specified, each entry implies a redefinition of the same storage area. The last clause in the FD entry must be immediately followed by a period and a space.

File-Name

The file-name must follow the level indicator, and must be the same as that specified in the associated file control entry.

The file-name must follow the rules of formation for a user-defined word; at least one character must be alphabetic. The file-name must be unique within this program.

BLOCK CONTAINS Clause

This clause is syntax-checked, but is treated as documentation except for tape files.

The BLOCK CONTAINS clause specifies the size of a physical record. When the BLOCK CONTAINS clause is omitted, the compiler assumes that records are not blocked. Thus, this clause can be omitted when each physical record contains only one complete logical record.

Format

$$\left[\text{BLOCK CONTAINS } \left[\text{integer-1 TO } \right] \text{integer-2 } \left\{ \begin{array}{l} \text{RECORDS} \\ \text{CHARACTERS} \end{array} \right\} \right]$$

Integer-1 and integer-2 must be nonzero unsigned integers.

When neither the CHARACTERS nor RECORDS phrase is specified, the CHARACTERS phrase is assumed.

RECORDS Phrase: When the RECORDS phrase is specified, the physical record size is the number of logical records contained in each physical record.

Note: Maximum record size is 32 767; maximum block size is 32 767. These maximums include any control bytes required for variable blocked records; thus, the maximum size data record for a variable-blocked record is 32 759.

CHARACTERS Phrase: When the CHARACTERS phrase is specified or implied, the physical record size is specified as the number of character positions required to store the physical record no matter what USAGE clause the characters within the data record have.

If only integer-2 is specified, it specifies the exact character size of the physical record. When integer-1 and integer-2 are both specified, they represent, respectively, the minimum and maximum character size of the physical record.

Note: Each variable record contains a 4-byte header and each block contains a 4-byte header when the data is transferred to tape. However, these 4-byte headers are provided by the system and are of no concern to

the COBOL user except that the maximum size of a variable record is restricted to 32 759.

When variable records are used, the BLOCK CONTAINS clause specifies the maximum physical record length, while the logical record length for each record is inferred by the compiler from the record name used in a WRITE statement. If an explicit length is required after a READ statement, the user can obtain it through the I-O-FEEDBACK mnemonic-name.

RECORD CONTAINS Clause

The RECORD CONTAINS clause specifies the size of a file's data records.

Format

[RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]

The RECORD CONTAINS clause is never required because the size of each record is completely defined in the record description entries. When this clause is specified, the following rules apply:

- Integer-3 and integer-4 must be unsigned, nonzero integers.
- When both integer-3 and integer-4 are specified, integer-3 specifies the size of the smallest data record, and integer-4 specifies the size of the largest data record.
- Integer-4 must not be specified alone unless all the records are the same size. If all records are the same size, integer-4 specifies the exact number of characters in the record.
- The record size must be specified as the number of character positions needed to store the record internally; that is, size is specified in terms of the number of bytes occupied internally by the record's characters, regardless of the number of characters used to represent the item within the record. The size of a record is determined according to the rules for obtaining the size of a group item. For a further description of record size, see "USAGE Clause" in this chapter.

Note: When the RECORD CONTAINS clause is omitted, the record lengths are determined by the compiler from the record descriptions. When one of the entries within a record description contains an OCCURS DEPENDING ON clause, the compiler uses the maximum value of the variable length item to calculate the record length.

Programming Note: The system supports variable length physical records only for files on tape. For all other files, the logical records are truncated or padded to the length of the record as defined in the CRTxxx F CL command. User length in the following table is defined as the largest record associated with the given file, as specified by its record description.

Input/Output Type	User Length Less Than File Record Length	User Length Greater Than File Record Length
Input	Truncation	Pad with blanks.
Output	Pad with blanks	Truncation if old file (non-empty); for new (empty files) the larger record length is used.

LABEL RECORDS Clause

The LABEL RECORDS clause specifies whether labels are present or omitted. The LABEL RECORDS clause is required in every FD entry. Format 3 (TAPEFILE) is the only format in which this clause is not treated as documentation.

Format

$$\underline{\text{LABEL}} \left\{ \begin{array}{l} \text{RECORD IS} \\ \text{RECORDS ARE} \end{array} \right\} \left\{ \begin{array}{l} \text{STANDARD} \\ \text{OMITTED} \end{array} \right\}$$

IBM Extension

The LABEL RECORDS clause can be changed at execution time by specifying the REELS parameter of the Override with Tape File (OVRTAPF) CL command. See the *CL Reference Manual* for more information on this command.

End of IBM Extension

STANDARD Phrase: The STANDARD phrase specifies that labels conforming to system specifications exist for this file. This phrase must be specified for files assigned to DISKETTE, DISK, and DATABASE. (See “FILE-CONTROL Paragraph” in Chapter 3.)

OMITTED Phrase: The OMITTED phrase specifies that no labels exist for this file. This phrase must be specified for files assigned to READER, PUNCHPRINT, PRINT, and PRINTER. (See “FILE-CONTROL Paragraph” in Chapter 3.)

VALUE OF Clause

The VALUE OF clause is syntax-checked, but is treated as documentation. It specifies the description of an item in the label records associated with this file.

Format

$$\left[\begin{array}{l} \text{VALUE OF } \text{user-name-1} \text{ IS } \left\{ \begin{array}{l} \text{data-name-1} \\ \text{literal-1} \end{array} \right\} \\ \text{, user-name-2 IS } \left\{ \begin{array}{l} \text{data-name-2} \\ \text{literal-2} \end{array} \right\} \dots \end{array} \right]$$

User-name: This name follows the rules for the formation of a user-defined word.

DATA RECORDS Clause

The DATA RECORDS clause specifies the names of data records associated with this file. The DATA RECORDS clause is never required.

Format

$$\left[\text{DATA } \left\{ \begin{array}{l} \text{RECORD IS} \\ \text{RECORDS ARE} \end{array} \right\} \text{data-name-3 } \left[\text{, data-name-4} \right] \dots \right]$$

Data-name-3 and data-name-4 are the names of data records and must have 01 level-number record descriptions that have the same name associated with them.

The specification of more than one data-name indicates that this file contains more than one type of data record. Two or more record descriptions for this file occupy the same storage area. These records need not have the same description or length. The order in which the data-names are listed is not significant.

LINAGE Clause

The LINAGE clause specifies the depth of a logical page in terms of the number of lines. This clause also optionally specifies the line number at which the footing area begins, as well as the top and bottom margins of the logical page.

At execution time, the printer file being used determines the physical page size. This information is used to issue appropriate space and eject commands to produce the logical page as defined in the LINAGE clause. Thus, the logical page can contain multiple physical pages, or one physical page can contain multiple logical pages.

Format

$$\left[\text{LINAGE IS } \left\{ \begin{array}{l} \text{data-name-5} \\ \text{integer-5} \end{array} \right\} \text{ LINES } \left[, \text{ WITH FOOTING AT } \left\{ \begin{array}{l} \text{data-name-6} \\ \text{integer-6} \end{array} \right\} \right] \right. \\ \left. \left[, \text{ LINES AT TOP } \left\{ \begin{array}{l} \text{data-name-7} \\ \text{integer-7} \end{array} \right\} \right] \left[, \text{ LINES AT BOTTOM } \left\{ \begin{array}{l} \text{data-name-8} \\ \text{integer-8} \end{array} \right\} \right] \right]$$

The LINAGE clause can be specified only for files assigned to the device PRINTER. See “FILE-CONTROL Paragraph” in Chapter 3.

All integers must be unsigned. All data-names must be described as unsigned integer data items.

LINAGE Integer-5/Data-Name-5: Integer-5 or the value in data-name-5 specifies the number of lines that can be written and/or spaced on this logical page. The area of the page that these lines represent is called the page body. The value must be greater than zero.

WITH FOOTING Phrase: Integer-6 or the value in data-name-6 specifies the first line number of the footing area within the page body. The footing line number must be greater than zero, but it must not be greater than the number for the last line of the page body. The footing area extends between those two lines. If this phrase is not specified, the assumed value is equal to that of the page body (integer-5 or data-name-5).

LINES AT TOP Phrase: Integer-7 or the value in data-name-7 specifies the number of lines in the top margin of the logical page. The value of integer-7 or data-name-7 can be zero. If this phrase is not specified, zero is assumed.

LINES AT BOTTOM Phrase: Integer-8 or the value in data-name-8 specifies the number of lines in the bottom margin of the logical page. The value of integer-8 or data-name-8 can be zero. If this phrase is not specified, zero is assumed.

Figure 4-1 illustrates the use of each phrase of the LINAGE clause.

LINAGE Clause Considerations: The logical page size specified in the LINAGE clause is the sum of all values specified in each phrase except the FOOTING phrase. If the LINES AT TOP and/or the LINES AT BOTTOM phrases are zero, each logical page immediately follows the preceding logical page with no additional spacing provided.

At the time an OPEN OUTPUT statement is executed, the values of integer-5, integer-6, integer-7, and integer-8 are used to determine the page body, first footing line, top margin, and bottom margin of the logical page for this file. These values are then used for all logical pages printed for this file during a given execution of the program.

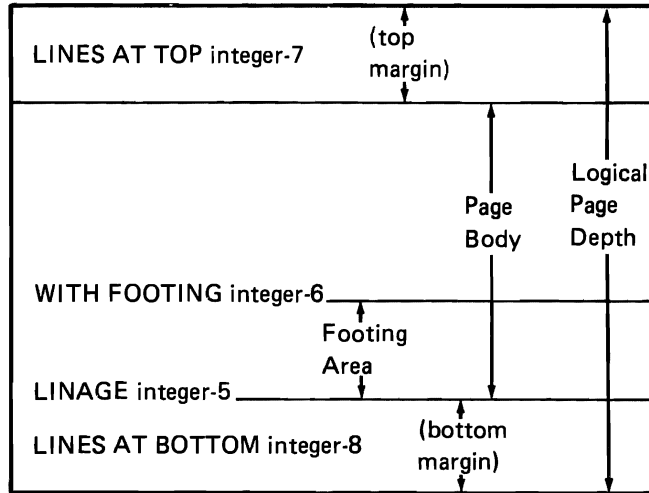


Figure 4-1. LINAGE Clause and Logical Page Depth

If the FOOTING phrase is specified and the value of data-name-6 or integer-6 is equal to the LINAGE value of data-name-5 or integer-5, one line (the last line of the logical page) is available for footing information. If the FOOTING phrase is not specified, no footing area is provided at the end of the logical page, even though the default FOOTING value is data-name-5 or integer-5.

Data-name-5, data-name-6, data-name-7, and data-name-8 cause the following actions to take place:

- Their values at the time an OPEN OUTPUT is executed are used to determine the page body, the first footing line, the top margin, and the bottom margin for the first logical page only.
- Their values at the time a WRITE ADVANCING statement causes page ejection are used to determine the page body, first footing line, top margin, and bottom margin for the next succeeding logical page only.

LINAGE-COUNTER Special Register: For each FD entry containing a LINAGE clause, a separate LINAGE-COUNTER special register is generated. LINAGE-COUNTER is initialized to one when an OPEN statement for this file is executed. LINAGE-COUNTER is automatically modified by any WRITE statement for this file.

If more than one FD has a LINAGE clause, then when LINAGE-COUNTER special register is referred to in the PROCEDURE DIVISION, the user must qualify each LINAGE-COUNTER with its related file-name. For example, LINAGE-COUNTER OF FILE-A.

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

CODE-SET Clause

The CODE-SET clause is valid only for files assigned to TAPEFILE or DISKETTE. This clause specifies the character code that is used to represent data on a magnetic tape file or diskette file.

Format

[CODE-SET IS alphabet-name]

When the CODE-SET clause is specified, the following rules apply:

- Alphabet-name identifies the character code convention that is used to represent data on the input/output device.
- All data in this file must have USAGE DISPLAY.
- If signed numeric data is present, it must be described by the SIGN IS SEPARATE clause.
- Alphabet-name must be defined in the SPECIAL-NAMES paragraph as STANDARD-1 for ASCII encoded files or as NATIVE for EBCDIC encoded files.

The CODE-SET clause specifies the algorithm for converting the character codes on the input/output medium from or to the internal EBCDIC character set.

IBM Extension

If the CODE-SET clause is omitted, the CODE parameter of the Create Diskette File (CRTDKTF) or the Create Tape File (CRTTAPF) CL command is used.

The CODE-SET clause can be changed at execution time by specifying the CODE parameter on the Override with Diskette File (OVRDKTF) or the Override with Tape File (OVRTAPF) CL command. See the *CL Reference Manual* for more information on these commands.

End of IBM Extension

Data Description

All the data used in a COBOL program is described using a uniform system of representation. The basic concepts of data description are discussed in this chapter, as well as the actual COBOL clauses used to describe data.

Data Description Concepts

Most of the data processed by a COBOL program is presented in hierarchically arranged records. This is necessary because most data must be divided into subdivisions for processing. To subdivide such records, COBOL uses a hierarchical concept of levels.

For example, in a department store's customer file, one complete record could contain all data pertaining to one customer. Subdivisions within that record could be: customer name, customer address, account number, department number of sale, unit amount of sale, dollar amount of sale, previous balance, and other pertinent information.

Level Concepts

Because records must be divided into logical subdivisions, the concept of levels is inherent in the structure of a record. Once a record has been subdivided, it can be further subdivided to provide more detailed data references.

The basic subdivisions of a record (that is, those fields that are not further subdivided) are called elementary items. Thus, a record can be made up of a series of elementary items, or it can itself be an elementary item.

It might be necessary to refer to a set of elementary items. Thus, elementary items can be combined into group items. Groups can be combined into a more inclusive group that contains two or more subgroups. Thus, within one hierarchy of data items, an elementary item can belong to more than one group item.

Level-Numbers

A system of level-numbers specifies the organization of elementary and group items into records. Special level-numbers are also used to identify data items used for special purposes.

Each group and elementary item in a record requires a separate entry, and each must be assigned a level-number. The following level-numbers are used to structure records:

- 01 This level-number specifies the record itself and is the most inclusive level-number possible. A level-01 entry can be either a group item or an elementary item.
- 02-49 These level-numbers specify group and elementary items within a record. Less inclusive data items are assigned higher (not necessarily consecutive) level-numbers.

A group item includes all group and elementary items following it until a level-number less than or equal to the level-number of this group is encountered.

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

IBM Extension

Elementary items or group items that are immediately subordinate to one group item can have unequal level-numbers. For example, group item A consists of items B, C, and D:

```
01  A
    05  B PIC X(4).
    04  C PIC X(20).
    02  D PIC 99.
```

IBM does not recommend such coding practices, and this extension is provided only for compatibility.

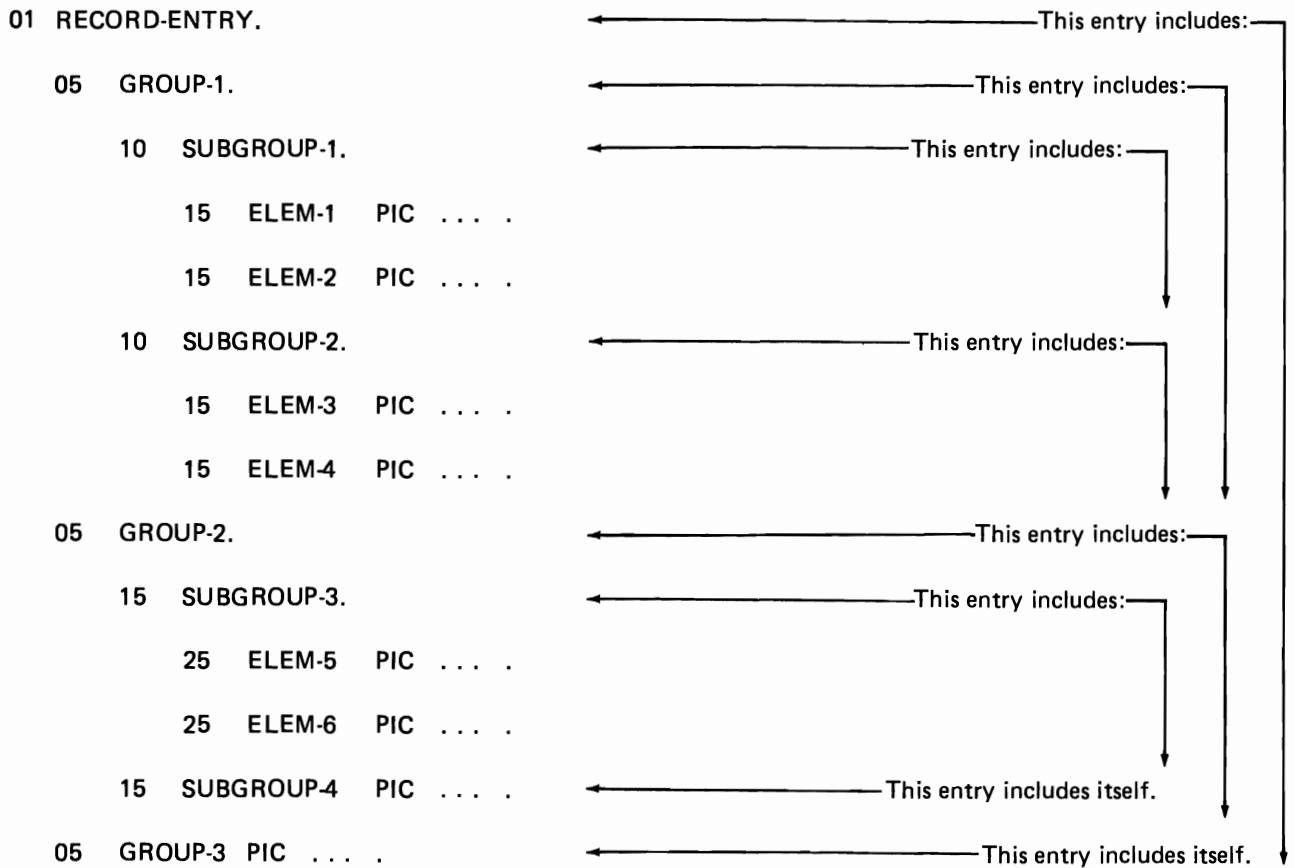
End of IBM Extension

Figure 4-2 illustrates the level-number concept. Notice that all groups immediately subordinate to the level-01 entry have the same level-number. Notice also that elementary items from different subgroups do not necessarily have the same level-number, and that elementary items can be specified at any level within the hierarchy. Figure 4-2 shows the COBOL record-description entry in the left portion of the figure; it shows the subdivision of the entry in the right portion of the figure.

Note: Level-numbers 01 through 09 can also be written as 1 through 9.

The COBOL record description entry is written as follows :

The items included in the hierarchy of each level are indicated below:



The storage arrangement is illustrated below:

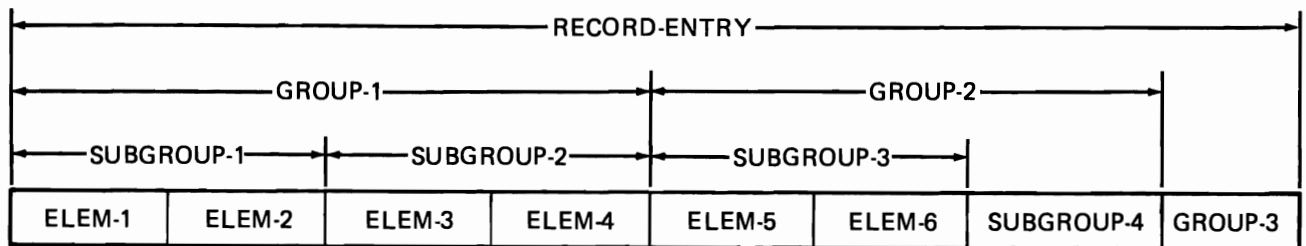


Figure 4-2. Level-Number Concepts

Special Level-Numbers

Special level-numbers identify items that do not structure a record. The following are special level-numbers:

- 66 This level-number identifies elementary or group items described by a RENAME clause. Such items regroup previously defined data items.
- 77 This level-number identifies independent data description entries in the Working-Storage or Linkage Section. These items are not subdivisions of other items, and are not themselves subdivided.
- 88 This level-number identifies any condition-name entry that is associated with a particular value or values of a conditional variable. An example is given under "VALUE Clause" in this chapter.

Note: Level-77 and level-01 entries in the Working-Storage Section and Linkage Section must be given unique data-names because neither can be qualified. If subordinate data-names can be qualified, they need not be unique.

Indentation

Successive data description entries can begin in the same column as preceding entries, or they can be indented according to level-number. Indentation is useful for documentation, but it does not affect the action of the compiler.

Classes of Data

All data used in a COBOL program can be divided into four classes and six categories. Every elementary item in a program belongs to one of the classes as well as one of the categories. Every group item belongs to the alphanumeric class even if the subordinate elementary items belong to another class and category. Figure 4-3 shows the relationship of data classes and categories.

IBM Extension

Boolean data is an IBM extension that provides a means of modifying and passing the values of the indicators associated with the display screen formats. A Boolean value of 0 is the off status of the indicator, and a Boolean value of 1 is the on status of the indicator.

A Boolean literal contains a single 0 or 1 and is enclosed in quotes and immediately preceded by an identifying B. The Boolean literal is defined as either B"0" or B"1". A Boolean character occupies 1 byte. The figurative constant ZERO can be used as a Boolean literal, and the reserved word ALL is valid with a Boolean literal.

End of IBM Extension

Level of Item	Class	Category
Elementary	Alphabetic Boolean Numeric Alphanumeric	Alphabetic Boolean Numeric Numeric edited Alphanumeric edited Alphanumeric
Group	Alphanumeric	Alphabetic Boolean Numeric Numeric edited Alphanumeric edited Alphanumeric

Figure 4-3. Classes and Categories of Data

Standard Alignment Rules

The standard alignment rules for positioning data in an elementary item depend on the data category of the receiving item (that is, the item into which the data is placed).

Numeric Items: When a numeric item is the receiving item, the following rules apply:

- The data is aligned on the assumed decimal point (PICTURE character V) and, if necessary, truncated or padded with zeros. (An assumed decimal point is one that has logical meaning but does not exist as a character in the data.)
- If an assumed decimal point is not explicitly specified, the receiving item is treated as though an assumed decimal point is specified immediately to the right of the field. The data is then treated as in the preceding rule.

Numeric Edited Items: The data is aligned on the decimal point and, if necessary, truncated or padded with zeros at either end, except when editing causes replacement of leading zeros.

Alphanumeric, Alphanumeric Edited, Alphabetic: For these data categories, the following rules apply:

- The data is aligned at the leftmost character position and, if necessary, truncated or padded with spaces at the right.
- If the JUSTIFIED clause is specified for this receiving item, the above rule is modified as described in the JUSTIFIED clause.

Standard Data Format

COBOL makes data description as machine independent as possible. For this reason, the properties of the data are described in a standard data format rather than a machine-oriented format.

The standard data format uses the decimal system to represent numbers no matter what base is used by the system. The nonnumeric data can contain any characters that are in the native character set, that is, nonnumeric data is not limited to just the COBOL character set or the nonnumeric COBOL characters.

Character-String and Item Size

In COBOL, the size of an elementary item is determined through the number of character positions specified in its PICTURE character-string. In storage, however, the size is determined by the actual number of bytes the item occupies as determined by the combination of its PICTURE character-string and its USAGE clause.

When an arithmetic item is moved from a longer field to a shorter one, the compiler truncates the data to the number of characters represented in the shorter item's PICTURE character-string.

For example, if a sending field with PICTURE S99999 and containing the value +12345 is moved to a COMPUTATIONAL receiving field with PICTURE S99, the data is truncated to +45.

Signed Data

There are two categories of algebraic signs used in COBOL: operational and editing.

Operational Signs

Operational signs (+, -) are associated with signed numeric items and indicate their algebraic properties. The internal representation of an algebraic sign depends on the item's USAGE clause and optionally upon its SIGN clause. Zero is considered a unique value regardless of the operational sign. An unsigned field is always assumed to be positive or zero.

Editing Signs

Editing signs are associated with numeric edited items. Editing signs are PICTURE symbols (+, -, CR, DB) that identify the sign of the item in edited output.

Record Description Entry

A record description entry consists of one or more data description entries. The maximum length of a record description entry is restricted to 32 767 bytes.

Data Description Entry

A data description entry specifies the characteristics of a particular data item. The maximum length for any item that is not otherwise restricted is 32 767 bytes. The general formats are:

Format 1

level-number { data-name-1
 FILLER }

[REDEFINES data-name-2]

[{ PICTURE } IS character-string
 { PIC }]

[[USAGE IS] { DISPLAY
 COMPUTATIONAL
 COMP
 COMPUTATIONAL-3
 COMP-3
 COMPUTATIONAL-4
 COMP-4
 INDEX }]

[[SIGN IS] { LEADING
 TRAILING } [SEPARATE CHARACTER]]

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

[{ ASCENDING
 DESCENDING } KEY IS data-name-4 [, data-name-5] . . .] . . .

[INDEXED BY index-name-1 [, index-name-2] . . .]]

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

[{ JUSTIFIED } RIGHT
 JUST]

[BLANK WHEN ZERO]

[VALUE IS literal] .

Format 2

66 data-name-1 RENAMES data-name-2 [{ THROUGH } data-name-3] .

Format 3

88 condition-name { VALUE IS } literal-1 [{ THROUGH } literal-2]
[literal-3 [{ THROUGH } literal-4]]

Format 4 – Boolean Data

```

level-number { data-name-1
              FILLER }

[ REDEFINES data-name-2 ]

[ { PICTURE
  PIC } IS 1 ]

[ USAGE IS DISPLAY ]

[ OCCURS { integer-1 TO integer-2 TIMES DEPENDING ON data-name-3
          integer-2 TIMES }

          [ INDEXED BY index-name-1 [ index-name-2 ] . . . ] ]

[ { INDICATOR
  INDICATORS } integer-3 ]

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

[ { JUSTIFIED } RIGHT ]

[ VALUE IS Boolean-literal ] .
    
```

Format 1

This format is used for record description entries in all sections and for level-77 entries in the Working-Storage and Linkage Sections. The following rules apply:

- Level-number can be any number from 01 through 49 or 77. Level-numbers from 01 through 09 can be coded as 1 through 9.
- The clauses can be written in any order, with two exceptions:
 - The data-name/FILLER clause must immediately follow the level-number.
 - When specified, the REDEFINES clause must immediately follow the data-name clause.
- The PICTURE clause must be specified for every elementary item except index data items.
- The BLANK WHEN ZERO, JUSTIFIED, PICTURE, and SYNCHRONIZED clauses are valid only for elementary items.
- Either a space, or a comma or a semicolon followed by a space, must separate clauses.
- Each entry must end with a period followed by a space.

Format 2

This format regroups previously defined items. The following rules apply:

- A level-66 entry cannot rename another level-66 entry, nor can it rename a level-01, level-77, or level-88 entry.
- All level-66 entries associated with one record must immediately follow the last data description entry in that record.
- The entry must end with a period followed by a space.

See “RENAMES Clause” later in this chapter for a further description.

Format 3

This format describes condition-names. A condition-name is a user-specified name that associates value(s) and/or a range(s) of values with a conditional variable.

A conditional variable is a data item that can assume one or more values that can, in turn, be associated with a condition-name. The following rules for condition-name entries apply:

- Any entry beginning with level-number 88 is a condition-name entry.

- The condition-name entries associated with a particular conditional-variable must immediately follow the conditional variable entry. The conditional variable can be any elementary data description entry except another condition-name, index data item, or level-66 entry.
- A condition-name can be associated with a group item data description entry. The following rules apply:
 - The condition-name value must be specified as a nonnumeric literal or figurative constant.
 - The size of the condition-name value must not exceed the sum of the sizes of all the elementary items within the group.
 - No element within the group may contain a JUSTIFIED or SYNCHRONIZED clause.
 - No USAGE other than USAGE IS DISPLAY may be specified within the group.
- Condition-names can be specified both at the group level and at subordinate levels within the group.
- The relation test implied by the definition of a condition-name at the group level is performed in accordance with the rules for comparison of nonnumeric operands regardless of the nature of elementary items within the group.
- Either a space or a comma or a semicolon followed by a space, must separate successive operands.
- Each entry must end with a period followed by a space.

Examples of both elementary and group condition-name entries are given under “VALUE Clause” in this chapter.

Format 4 – Boolean Data

See Chapter 7 for a discussion of this format.

Level-Numbers

The level-number specifies the hierarchy of data within a record and also identifies special-purpose data entries.

Format

level-number

The following rules for level-numbers apply:

- A level-number begins a data description entry, a regrouped item, or a condition-name entry.
- Level-numbers 01 and 77 must begin in Area A.

- Level-numbers 02-49, 66, and 88 can begin in either Area A or Area B and must be followed by a space.
- Single-digit level-numbers 1 through 9 can be substituted for level-numbers 01 through 09.

Data-Name or FILLER Clause

A data-name explicitly identifies the data being described; the keyword FILLER specifies an item that is never explicitly referenced in the program.

Format

data-name
FILLER

In a data description entry, either the data-name or the keyword FILLER must be the first word following the level-number. The data-name identifies a data item by referring to the field, not to a particular value. This data item can assume a number of different values during the course of a program.

A data-name can begin anywhere in Area B. A data-name must contain at least one alphabetic character.

Entries at level-numbers 01 and 77 in the Working-Storage and Linkage Sections cannot be qualified, and therefore require unique data-names. Subordinate data-names that can be qualified do not require unique data-names.

The keyword FILLER specifies an elementary item in a record that is never explicitly referred to. The word FILLER can be written anywhere in Area B.

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

IBM Extension

A FILLER item can be used as a group item definition. Subordinate data items can then be referenced by the appropriate data-name.

End of IBM Extension

REDEFINES Clause

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

Format

level-number data-name-1 REDEFINES data-name-2

Level-number and data-name-1 are not part of the REDEFINES clause itself, and are included in the format only for clarity.

If specified, the REDEFINES clause must be the first entry following data-name-1.

The level-number of data-name-1 and data-name-2 must be identical and must not be level-66 or level-88.

Data-name-1 is the redefining item and is an alternative description for the data-name-2 area.

Data-name-2 is the redefined item.

Implicit redefinition is assumed when more than one level-01 entry subordinate to an FD entry is written. In such level-01 entries, the REDEFINES clause must not be specified.

Redefinition begins at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered. No entry having a level-number numerically lower than those of data-name-1 and data-name-2 can occur between these entries.

In the following example, A is data-name-2, and B is data-name-1. Redefinition begins with B and includes the two subordinate items B-1 and B-2. Redefinition ends when the level-05 item C is encountered.

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

The data description entry for data-name-2 cannot contain a REDEFINES clause or an OCCURS clause. However, data-name-2 can itself be subordinate to an item that contains either clause. If data-name-2 is subordinate to an OCCURS clause, it must not be subscripted or indexed in the REDEFINES clause.

The redefined item, the redefining item, and any items subordinate to them cannot contain an OCCURS DEPENDING ON clause.

When data-name-1 is specified with a level-number other than 01, it must specify a storage area of the same size as data-name-2.

Multiple redefinitions of the same storage area are permitted. The entries giving the new descriptions of the storage area must immediately follow the description of the redefined area without intervening entries that define new character positions. Multiple redefinitions must all use the data-name of the original entry that defined this storage area. For example:

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

Data-name-1 and any subordinate entries must not contain any VALUE clauses. This rule does not apply to condition-name entries.

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

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

Data items can also be rearranged within an area. For example:

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

When an area is redefined, all descriptions of the area are always in effect; that is, redefinition does not cause any data to be erased and does not supersede the previous description. Thus, if B REDEFINES A has been specified, either of the two procedural statements MOVE X TO B and MOVE Y TO A could be executed at any point in the program.

In the first case, the area described as B would assume the value of X. In the second case, the same physical area (described now as A) would assume the value of Y. If the second statement is executed immediately after the first, the value of Y replaces the value of X in the one storage area.

The USAGE clause of a redefining data item need not be the same as that of a redefined item. This does not, however, cause any change in existing data. For example:

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

The bit configuration of the DISPLAY value 8 is 1111 0000 1111 1000. Redefining B does not change the bit configuration of the data in the storage area. Therefore, the two statements, ADD B TO A and ADD C TO A give different results. In the first case, the value 8 is added to A (because B has USAGE DISPLAY). In the second statement, the value -48 is added to A (because C has USAGE COMPUTATIONAL-4), and the bit configuration (truncated to 2 decimal digits) in the storage area has the binary value -48.

Unexpected results can occur when a redefining item is moved to a redefined item (that is, if B REDEFINES C and the statement MOVE B TO C is executed). Unexpected results can also occur when a redefined item is moved to a redefining item (from the previous example, unexpected results occur if the statement MOVE C TO B is executed).

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

```

05 REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 GRADE PICTURE X(4).
   10 SEMI-MONTHLY-PAY PICTURE
      999V99.
   10 WEEKLY-PAY REDEFINES
      SEMI-MONTHLY-PAY
      PICTURE 999V999.

05 TEMPORARY-EMPLOYEE REDEFINES
   REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 FILLER PICTURE X(6).
   10 HOURLY-PAY PICTURE 99V99.

```

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

```

05 REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 GRADE PICTURE X(4).
   10 SEMI-MONTHLY-PAY
      PICTURE 999V999.

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

```

USAGE Clause

The USAGE clause specifies the format of a data item in storage. The USAGE clause can be specified for an entry at any level. However, if it is specified at the group level, it applies to each elementary item in the group. The usage of an elementary item cannot contradict the explicit usage of a group to which the elementary item belongs.

The USAGE clause specifies the format in which data is represented in storage. Consideration must be given to how the data is used in the Procedure Division.

Format

[[USAGE IS	{	DISPLAY COMPUTATIONAL COMP COMPUTATIONAL-3 COMP-3 COMPUTATIONAL-4 COMP-4 INDEX	}]]
---	---	----------	---	---	---	---	---

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

DISPLAY Phrase

The DISPLAY phrase can be explicit or implicit. It specifies that the data item is stored in character form, one character per 8-bit byte. This corresponds to the form in which information is represented for keyboard input or for printed output. USAGE IS DISPLAY is valid for the following types of items:

- Alphabetic
- Alphanumeric
- Alphanumeric edited
- Numeric edited
- Boolean
- Zoned decimal (numeric).

Alphabetic, alphanumeric, alphanumeric edited, numeric edited, and Boolean items are discussed under “PICTURE Clause” later in this chapter.

Zoned Decimal Items: These items are sometimes referred to as external decimal items. Each digit of a number is presented by a single byte. The four high-order bits of each byte are zone bits; the four high-order bits of the low-order byte represent the sign of the item. If the number is positive, these four bits contain a hexadecimal F. If the number is negative, these four bits contain a hexadecimal D. The four low-order bits of each byte contain the value of the digit. When zoned decimal items are used for computations, the compiler performs the necessary conversions. The maximum length of a zoned decimal item is 18 digits.

The PICTURE character-string of a zoned item can contain only 9s, the operational sign symbol S, the assumed decimal point V, and one or more Ps.

Examples of zoned decimal items are shown in Figure 4-4.

Computational Phrases

The term computational refers to the following phrases of the USAGE clause:

- COMPUTATIONAL or COMP (packed decimal)

IBM Extension

- COMPUTATIONAL-3 or COMP-3 (packed decimal).
- COMPUTATIONAL-4 or COMP-4 (binary).

End of IBM Extension

A computational item represents a value to be used in arithmetic operations and must be numeric. If the USAGE of a group item is described with any of these options, it is the elementary items within the group that have this usage. The group itself is considered nonnumeric and cannot be used in numeric operations. The maximum length of a computational item is 18 decimal digits.

The PICTURE of a computational item can contain only:

9 (one or more numeric character positions)

S (one operational sign)

V (one implied decimal point)

P (one or more decimal scaling positions).

The COMPUTATIONAL phrase is specified for packed decimal items. Such an item appears in storage as 2 digits per byte, with the sign contained in the 4 rightmost bits of the rightmost byte. A packed decimal item can contain any of the digits 0 through 9 plus a sign. If the PICTURE of a packed decimal item does not contain an S, the sign position is occupied by a bit configuration that is interpreted as positive.

IBM Extension

The COMPUTATIONAL-3 phrase is specified for packed decimal items and is considered by the compiler to be equivalent to the COMPUTATIONAL phrase.

The COMPUTATIONAL-4 phrase is specified for binary data items. Such items have decimal equivalents consisting of the decimal digits 0 through 9, plus a sign.

The amount of storage occupied by a binary data item depends on the number of decimal digits defined in its PICTURE clause:

Digits in PICTURE Clause	Storage Occupied
1 through 4	2 bytes
5 through 9	4 bytes
10 through 18	8 bytes

The leftmost bit of the storage area is the operational sign.

End of IBM Extension

Examples of packed decimal and binary items are shown in Figure 4-5.

INDEX Phrase

The `USAGE IS INDEX` clause specifies that the data item named has an indexed format and, therefore, is an index data item. The index data item is an elementary item that can be used to save index-name values for future reference.

The `USAGE IS INDEX` clause is described in detail under “**TABLE HANDLING**” in Chapter 6.

Item	Description	Value	Internal Representation*	
Zoned Decimal	PIC S9999 DISPLAY	+1234	F1 F2 F3 F4	
		-1234	F1 F2 F3 D4	
		1234	F1 F2 F3 F4	
	PIC 9999 DISPLAY	+1234	F1 F2 F3 F4	
		-1234	F1 F2 F3 F4	
		1234	F1 F2 F3 F4	
	PIC S9999 DISPLAY SIGN LEADING	+1234	F1 F2 F3 F4	
		-1234	D1 F2 F3 F4	
		1234	F1 F2 F3 F4	
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+1234	F1 F2 F3 F4 4E	
		-1234	F1 F2 F3 F4 60	
		1234	F1 F2 F3 F4 4E	
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+1234	4E F1 F2 F3 F4	
		-1234	60 F1 F2 F3 F4	
		1234	4E F1 F2 F3 F4	
	Packed Decimal	PIC S9999 { COMP COMP-3 }	+1234	01 23 4F
			-1234	01 23 4D
		PIC 9999 { COMP COMP-3 }	+1234	01 23 4F
-1234			01 23 4F	
Binary		PIC S9999 COMP-4	+1234	04 D2
			-1234	FB 2E
	PIC 9999 COMP-4	+1234	04 D2	
		-1234	04 D2	
*The internal representation of each byte is shown as two hex digits. The bit configuration for each digit is as follows:				
Hex Digit	Bit Configuration	Hex Digit	Bit Configuration	
0	0000	8	1000	
1	0001	9	1001	
2	0010	A	1010	
3	0011	B	1011	
4	0100	C	1100	
5	0101	D	1101	
6	0110	E	1110	
7	0111	F	1111	
<i>Notes:</i>				
1. The leftmost bit of a binary number represents the sign: 0 is positive, 1 is negative.				
2. Negative binary numbers are represented in twos complement form.				
3. Hex 4E represents the EBCDIC character +. Hex 60 represents the EBCDIC character -.				
4. Specification of SIGN TRAILING (without the SEPARATE CHARACTER option) is the equivalent of the standard action of the compiler.				

Figure 4-4. Internal Representation of Numeric Items

SIGN Clause

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

Format

$$\left[\left[\text{SIGN IS} \right] \left\{ \begin{array}{l} \text{LEADING} \\ \text{TRAILING} \end{array} \right\} \left[\text{SEPARATE CHARACTER} \right] \right]$$

The SIGN clause can be specified only for a signed numeric data description entry (that is, one whose PICTURE character-string contains an S), or for a group item that contains at least one such elementary entry. USAGE IS DISPLAY must be specified either explicitly or implicitly.

Only one SIGN clause can apply to any one data description entry. The SIGN clause is required only when an explicit description of the properties and/or position of the operational sign is necessary.

The SIGN clause defines the position and mode of representation of the operational sign for the numeric data description entry to which it applies, or for each signed numeric data description entry subordinate to the group to which it applies.

If the SEPARATE CHARACTER phrase is not specified, then:

- The operational sign is presumed to be associated with the leading or trailing digit position (whichever is specified) of the elementary numeric data item.
- The character S in the PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).

If the SEPARATE CHARACTER phrase is specified, then:

- The operational sign is presumed to be the leading or trailing character position (whichever is specified) of the elementary numeric data item. This character position is not a digit position.
- The character S in the PICTURE character string is counted in determining the size of the data item (in terms of standard data format characters).
- + is the character used for the positive operational sign.
- - is the character used for the negative operational sign.

Every numeric data description entry whose PICTURE contains the symbol S is a signed numeric data description entry. If the SIGN clause is also

specified for such an entry and conversion is necessary for computations or comparisons, the conversion takes place automatically.

If no SIGN clause is specified for a signed numeric data description entry, the position and mode of representation for the operational sign is determined as explained in the USAGE clause description.

OCCURS Clause

The OCCURS clause specifies tables whose elements can be referred to by indexing or subscripting. This clause is described under “Data Division – Table Handling” in Chapter 6.

INDICATOR Clause

The INDICATOR clause is discussed under “Data Description Entry – Boolean Data” in Chapter 7.

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on a proper boundary in storage.

The SYNCHRONIZED clause is syntax-checked, but is treated as documentation for all items.

Format

$$\left[\left\{ \begin{array}{l} \text{SYNCHRONIZED} \\ \text{SYNC} \end{array} \right\} \left[\begin{array}{l} \text{LEFT} \\ \text{RIGHT} \end{array} \right] \right]$$

JUSTIFIED Clause

The JUSTIFIED clause overrides standard positioning rules for a receiving item of the alphabetic or alphanumeric categories.

Format

$$\left[\left\{ \begin{array}{l} \text{JUSTIFIED} \\ \text{JUST} \end{array} \right\} \text{RIGHT} \right]$$

The JUSTIFIED clause can be specified only at the elementary level. JUST is an abbreviation for JUSTIFIED and has the same meaning.

The JUSTIFIED clause must not be specified for a numeric item or for any item for which editing is specified. The JUSTIFIED clause must not be specified with level-66 (RENAMES) or level-88 (condition-name) entries.

IBM Extension
The JUSTIFIED clause can be specified for an alphanumeric edited item.
End of IBM Extension

When the JUSTIFIED clause is specified for a receiving item, the data is aligned at the rightmost character position in the receiving item, and:

- If the sending item is larger than the receiving item, the leftmost characters are truncated.
- If the sending item is smaller than the receiving item, the unused character positions at the left are filled with spaces.

When the JUSTIFIED clause is omitted, the rules for standard alignment are followed.

The following shows the difference between standard and justified alignment when the receiving field has a length of 5 character positions:

Alignment	Sending Field Value	Receiving Field Value
Standard	THE	THE
Justified right	THE	THE
Standard	TOO BIG	TOO B
Justified right	TOO BIG	BIG

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause specifies that an item is to be filled entirely with spaces when its value is zero. When the data item receives a value of zero through an explicit reference at execution time, it is set to blanks.

Format

[BLANK WHEN ZERO]

The BLANK WHEN ZERO clause can be specified only for elementary numeric or numeric edited items. When it is specified for a numeric item, the item is considered to be a numeric edited item.

If the BLANK WHEN ZERO clause is specified, the item contains nothing but spaces when its value is zero.

The BLANK WHEN ZERO clause must not be specified for level-66 or level-88 items.

The BLANK WHEN ZERO clause and the asterisk (*) suppression symbol must not be specified for the same entry.

VALUE Clause

The VALUE clause specifies the initial contents of a data item, or the value(s) associated with a condition-name. The two formats for the VALUE clause are as follows:

Format 1

[VALUE IS literal]

Format 2

88 condition-name { VALUE IS
VALUES ARE } literal-1 [{ THROUGH
THRU } literal-2]
[literal-3 [{ THROUGH
THRU } literal-4]]

Level-number 88 and condition-name are not part of the Format 2 VALUE clause itself, and are included in the format only for clarity. The use of the VALUE clause differs with the Data Division section in which it is specified.

File and Linkage Sections: The VALUE clause can only be used in condition-name entries.

Working-Storage Section: The VALUE clause is used in condition-name entries. It is also used to specify the initial value of any data item; the item assumes the specified value at the beginning of program execution. If the initial value is not explicitly specified, it is unpredictable.

General Considerations

The keywords THRU and THROUGH are equivalent.

The VALUE clause must not be specified for any item whose length is variable; that is, it is a group item that has an OCCURS DEPENDING ON clause subordinate to it.

For group entries, the VALUE clause must not be specified if the entry or an entry subordinate to it contains any of the following clauses: JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE DISPLAY).

The VALUE clause must not conflict with other clauses in the data description entry or in the data description of this entry's hierarchy. The following rules apply:

- Wherever a literal is specified, a figurative constant can be substituted.
- If the item is numeric, all VALUE clause literals must be numeric literals. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves with one additional restriction: the literal must not have a value that requires truncation of nonzero digits. If the literal is signed, the associated PICTURE character-string must contain a sign symbol (S).
- All numeric literals in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause for that item. For example, for PICTURE 99PPP, the literal must be zero or within the range 1000 through 99000. For PICTURE PPP99, the literal must be within the range .00000 through .00099.
- If the item is an alphabetic, alphanumeric, alphanumeric edited, or numeric edited item, all VALUE clause literals must be nonnumeric literals. The number of characters in the literal must not exceed the size of the item.

IBM Extension

- If the item is Boolean, the VALUE clause literal must be a Boolean literal.

End of IBM Extension

- The functions of the editing characters in a PICTURE clause are ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item. Therefore, any editing character must be included in the literal. For example, if the item is defined as PICTURE +999.99 and the value is to be +12.34, then the VALUE clause should be specified as VALUE "+012.34".
- A maximum of 32 767 bytes can be initialized by means of a single VALUE clause.

Format 1 Considerations

This format specifies the initial value of a data item in storage. Initialization is independent of any BLANK WHEN ZERO or JUSTIFIED clause specified.

A Format 1 VALUE clause must not be specified for an entry that contains or is subordinate to an entry that contains a REDEFINES or OCCURS clause.

If the VALUE clause is specified at the group level, the literal must be a nonnumeric literal or a figurative constant. The group area is initialized without consideration for the subordinate entries within this group. In addition, the VALUE clause must not be specified for subordinate entries within this group.

Format 2 Considerations

This format associates a value, values, and/or range(s) of values with a condition-name. Each such condition-name requires a separate level-88 entry.

The VALUE clause is required in a condition-name entry and must be the only clause in the entry. Each condition-name entry is associated with a preceding conditional variable. Thus, every level-88 entry must always be preceded either by the entry for the conditional variable or by another level-88 entry when several condition-names apply to one conditional variable. Such level-88 entries implicitly have the PICTURE characteristics of the conditional variable.

Every condition-name can be qualified by the name of its associated conditional variable and by the qualifier(s) of the conditional variable. If the associated conditional variable requires subscripts or indexes, each procedural reference to the condition-name must be subscripted or indexed as required for the conditional variable.

When only literal-1 is specified, the condition-name is associated with a single value.

When literal-1, literal-3 and so on are specified, the condition-name is associated with several single values.

When literal-1 THRU literal-2 is specified, the condition-name is associated with one range of values.

- If the literals are numeric, literal-1 must be less than literal-2.
- If the literals are nonnumeric, literal-1 must appear before literal-2 in the program collating sequence.

When literal-1 THRU literal-2, literal-3 THRU literal-4, and so on are specified, the condition-name is associated with more than one range of values.

- If the literals are numeric, literal-1 must be less than literal-2, literal-3 must be less than literal-4, and so on.

- If the literals are nonnumeric, literal-1 must appear before literal-2 in the program collating sequence, literal-3 must appear before literal-4 in the program collating sequence, and so on.

One or more single values and one or more ranges of values can be specified in a single Format 2 VALUE clause.

The type of literal in a condition-name entry must be consistent with the data type of the conditional variable. In the following example, CITY-COUNTY-INFO, COUNTY-NO, and CITY are conditional variables; the associated condition-names immediately follow the level-number 88. The PICTURE associated with COUNTY-NO limits the condition-name value to a 2-digit numeric literal. The PICTURE associated with CITY limits the condition-name value to a 3-character nonnumeric literal. Any values for the condition-names associated with CITY-COUNTY-INFO cannot exceed 5 characters, and the literal (because this is a group item) must be nonnumeric:

```

05 CITY-COUNTY-INFO.
   88 BRONX                VALUE "03NYC".
   88 BROOKLYN             VALUE "24NYC".
   88 MANHATTAN            VALUE "31NYC".
   88 QUEENS                VALUE "41NYC".
   88 STATEN-ISLAND        VALUE "43NYC".
10 COUNTY-NO                PICTURE 99.
   88 DUTCHESS              VALUE 14.
   88 KINGS                 VALUE 24.
   88 NEW-YORK              VALUE 31.
   88 RICHMOND              VALUE 43.
10 CITY                      PICTURE X(3).
   88 BUFFALO               VALUE "BUF".
   88 NEW-YORK-CITY         VALUE "NYC".
   88 POUGHKEEPSIE         VALUE "POK".
05 POPULATION...

```

The following example shows the use of the THRU phrase. In this example, the number of miles a person drives to work each day is categorized.

```

05 MILEAGE PIC 9(2)V9.
   88 LOW VALUE 0 THRU 09.9.
   88 MED VALUE 10.0 THRU 19.9.
   88 HIGH VALUE 20.0 THRU 99.9.

```

Condition-names are used procedurally in condition-name conditions, and are described under "Conditional Expressions" in Chapter 5.

PICTURE Clause

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

Format

$$\left[\left\{ \begin{array}{l} \text{PICTURE} \\ \text{PIC} \end{array} \right\} \text{ IS character-string} \right]$$

The PICTURE clause must be specified for every elementary item except an indexed data item. The PICTURE clause can be specified only at the elementary level. PIC is an abbreviation for PICTURE and has the same meaning.

The character-string is made up of certain COBOL characters used as symbols. The allowable combinations determine the category of the data item. The character-string can contain a maximum of 30 characters.

Symbols Used in the PICTURE Clause

The functions of each PICTURE clause symbol are defined in the following list. Any punctuation character appearing within the PICTURE character-string is not considered a punctuation character, but rather as a PICTURE character-string symbol.

- A Each A in the character-string represents a character position that can contain only a letter of the alphabet or a space.
- B Each B in the character-string represents a character position into which the space character will be inserted.
- P The P indicates an assumed decimal scaling position, and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or in items that appear as arithmetic operands. In any operation converting data from one form of internal representation to another, if the item being converted is described with the PICTURE symbol P, each digit position described by a P is considered to contain the value zero, and the size of the item is considered to include these zero digit positions.

For example, PICTURE PPP99 DISPLAY defines a 2-character item whose value is zero or ranges from .00001 through .00099. PICTURE 99PPP DISPLAY defines a 2-character item whose value is zero or ranges from 1000 through 99000.

The scaling position character P can appear only to the left or right of the other characters in the string as a continuous string of Ps within a PICTURE description. The sign character S and the assumed decimal point V are the only characters which can appear to the left of a leftmost string of Ps. Because the scaling position character P implies an assumed decimal point (to the left of the Ps if the Ps are leftmost PICTURE characters; to the right of the Ps if the Ps are rightmost PICTURE characters), the assumed decimal point symbol V is redundant as either the leftmost or rightmost character within such a PICTURE description.

- S The symbol S is used in a PICTURE character-string to indicate the presence (but not the representation or, necessarily, the position) of an operational sign. The sign must be written as the leftmost character in the PICTURE string. An operational sign indicates

whether the value of an item involved in an operation is positive or negative. The symbol S is not counted in determining the size of the elementary item, unless an associated SIGN clause specifies the SEPARATE CHARACTER option.

- V The V is used in a character-string to indicate the location of the assumed decimal point and can appear only once in a character-string. The V does not represent a character position and, therefore, is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the V should not be included in this PICTURE string.
- X Each X in the character-string represents a character position that can contain any allowable character from the EBCDIC set.
- Z Each Z in the character-string represents a leading numeric character position. When that position contains a zero, the zero is replaced by a space character. Each Z is counted in the size of the item.
- 9 Each 9 in the character-string represents a character position that contains a numeral and is counted in the size of the item.

IBM Extension

- 1 The character 1 represents a character position that contains a Boolean value of B"1" or B"0". Usage must be explicitly or implicitly defined as DISPLAY.

End of IBM Extension

- 0 Each zero in the character-string represents a character position into which the numeral zero will be inserted. Each zero is counted in the size of the item.
 - / Each slash in the character-string represents a character position into which the slash character will be inserted. Each slash is counted in the size of the item.
 - ,
 - .
- Each comma in the character-string represents a character position into which a comma will be inserted. This character is counted in the size of the item. The comma insertion character cannot be the last character in the PICTURE character-string.
- When a period appears in the character-string, it is an editing symbol that represents the decimal point for alignment purposes. In addition, it represents a character position into which a period will be inserted. This character is counted in the size of the item. The period insertion character cannot be the last character in the PICTURE character-string.

Note: For a given program, the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange, the rules for the period

apply to the comma, and the rules for the comma apply to the period wherever they appear in a PICTURE clause.

+, -, CR, DB

These symbols are used as editing sign control symbols. Each symbol represents the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in one character-string. Each character used in the symbol is counted in determining the size of the data item.

- * Each asterisk (check protect symbol) in the character-string represents a leading numeric character position into which an asterisk will be placed when that position contains a zero. Each asterisk is counted in the size of the item.

The asterisk used as the zero suppression symbol and the BLANK WHEN ZERO clause must not appear in the same entry.

- 'cs' The currency symbol in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented either by the symbol \$ or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. The currency symbol is counted in the size of the item.

Note: Because the currency symbol can be replaced in the CURRENCY SIGN clause, the term 'cs' is used throughout this book rather than the actual currency symbol (\$).

Figure 4-5 gives the order in which PICTURE clause symbols must be specified.

Character-String Representation: The following symbols can appear more than once in one PICTURE character-string:

A B P X Z 9 0 / , + - * 'cs'

Each time one of these symbols appears in the character-string, it represents an occurrence of that character or set of allowable characters in the data item.

First Symbol Second Symbol	Non-Floating Insertion Symbols										Floating Insertion Symbols					Other Symbols								
	B	0	/	,	.	{+} ¹	{-} ¹	{CR}{DB}	'cs' ²	{Z}* ¹	{Z}* ¹	{+} ¹	{-} ¹	'cs' ²	'cs' ²	9	A X	S	V	P ¹	P ¹	1		
Non-Floating Insertion Symbols	B	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		
	{CR}{DB}	x	x	x	x	x				x	*	x			x	x	x			x	x	x		
'cs'						x																		
Floating Insertion Symbols	{Z}* ¹	x	x	x	x		x		x	x														
	{Z}* ¹	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		
	'cs'	x	x	x	x		x							x										
	P	x	x	x	x	x	x								x	x				x		x		
Other Symbols	9	x	x	x	x	x	x			x	x		x			x	x	x	x		x			
	A X	x	x	x												x	x							
	S																							
	V	x	x	x	x		x			x	x		x			x			x		x			
	P	x	x	x	x		x			x	x		x			x			x		x			
	P						x			x									x	x		x		
	1																							

¹ Non-floating insertion symbols + and -, floating insertion symbols Z, *, +, -, and 'cs', and other symbol P appear twice in the above table. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the table represents its use to the right of the decimal point position.

² \$ is the default value for the currency symbol. This value can be replaced by a character specified in the currency SIGN clause.

An X at an intersection indicates that the symbol(s) at the top of the column can, in a given character-string, appear anywhere to the left of the symbol(s) at the left of the row.

Braces ({ }) indicate items that are mutually exclusive.

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

Figure 4-5. PICTURE Clause Symbol Order

An integer enclosed in parentheses immediately following any of these symbols specifies the number of consecutive occurrences of that symbol. The number of consecutive occurrences cannot exceed 32 767.

For example, the following two PICTURE clause specifications are equivalent:

```
PICTURE IS $99999.99CR
PICTURE IS $9(5).99CR
```

The following five symbols can each appear only once in one PICTURE character-string:

```
S V . CR DB 1
```

Data Categories and PICTURE Considerations: The allowable combinations of PICTURE symbols determine the data category of the item. Rules for each category follow.

The following rules apply for alphabetic items:

- The PICTURE character-string can contain only the symbols A and B.
- The contents of the item in standard data format must consist of any of the 26 letters of the alphabet and the space character.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal.

IBM Extension

Boolean items – the following rules apply:

- The PICTURE character-string can contain only the symbol 1.
- Only one character 1 can be specified.
- The USAGE of an item can only be DISPLAY.
- An associated VALUE clause must specify a Boolean literal (B"1" or B"0") or zero.
- The following clauses cannot be specified for a Boolean item:
 - SIGN clause
 - BLANK WHEN ZERO clause
 - ASCENDING/DESCENDING KEY clause.
- The INDICATOR clause can be specified (see "Indicators" in Chapter 7).

End of IBM Extension

The following rules apply for numeric items:

- The PICTURE character-string can contain only the symbols 9, P, S, and V.
- The number of digit positions must range from 1 through 18.
- The contents of a numeric item must be a combination of the digits 0 through 9. The numeric item can contain an operational sign. If the PICTURE contains an S, the contents of the item are treated as a positive or negative value, depending on the operational sign present in the data. If the PICTURE does not contain an S, the contents of the item are treated as an absolute value.
- If a VALUE clause is specified for an elementary numeric item, the literal must be numeric. If a VALUE clause is specified for a group item consisting of elementary numeric items, the group is considered alphanumeric, and the literal must therefore be nonnumeric.
- The USAGE clause of the item can be DISPLAY or COMPUTATIONAL.

IBM Extension
The USAGE can be COMPUTATIONAL-3 or COMPUTATIONAL-4.
End of IBM Extension

Examples of numeric items are shown in Figure 4-6.

PICTURE	Valid Range of Values
9999	0 through 9999
S99	-99 through +99
S999V9	-999.9 through +999.9
PPP999	0 through .000999
S999PPP	-1000 through -999000 and +1000 through +999000 or zero

Figure 4-6. Examples of Numeric Items

The following rules apply to alphanumeric items:

- The PICTURE character-string must consist of either:
 - The symbol X entirely.
 - Combinations of the symbols A, X, and 9. The item is treated as if the character-string contained only the symbol X. A PICTURE character-string containing all A's or all 9's does not define an alphanumeric item.
- The contents of the item in standard data format may be any allowable characters from the EBCDIC character set.
- USAGE DISPLAY must be either specified or implied.

- Any associated VALUE clause must specify a nonnumeric literal.

The following rules apply to alphanumeric edited items:

- The PICTURE character-string can contain the symbols:

A X 9 B 0 /

- The string must contain at least one of the following combinations:
 - At least one B and at least one X
 - At least one 0 and at least one X
 - At least one X and at least one /
 - At least one A and at least one 0
 - At least one A and at least one /.
- The contents of the item in standard data format can be any allowable character from the EBCDIC character set.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal. The literal is treated exactly as specified; no editing is performed.
- Alphanumeric edited items are subject to only one type of editing – simple insertion using the symbols 0, B, and /.

The following rules apply to numeric edited items:

- The PICTURE character-string can contain the following symbols:

B P V Z 9 0 / , . + - CR DB * 'cs'

The combinations of symbols allowed are determined from the PICTURE clause symbol order allowed (see Figure 4-4), and the editing rules (see the following section). The following additional rules also apply:

- The string must contain at least one of the following symbols:
 - B / Z 0 , . * + - CR DB 'cs'
- The number of digit positions represented in the character-string must be in the range of 1 through 18 inclusive.
- The total number of character positions in the string (including editing characters) must not exceed 30.
- The contents of those character positions representing digits in standard data format must be one of the digits 0 through 9.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal. The literal is treated exactly as specified; no editing is performed.

PICTURE Clause Editing

There are two general methods of performing editing in a PICTURE clause: by insertion, or by suppression and replacement.

There are four types of insertion editing: simple insertion, special insertion, fixed insertion, and floating insertion. There are two types of suppression and replacement editing: zero suppression and replacement with asterisks, and zero suppression and replacement with spaces.

The type of editing allowed for an item depends on its data category. The type of editing that is valid for each category is shown in Figure 4-7.

Each type of editing is discussed in detail in the following paragraphs.

Simple Insertion Editing: This type of editing is valid for alphabetic, alphanumeric edited, and numeric edited items. The valid insertion symbols for each category are shown in Figure 4-8.

Each insertion symbol is counted in the size of the item, and represents the position within the item where the equivalent characters will be inserted. Examples of simple insertion editing are shown in Figure 4-9.

Category	Type of Editing
Alphabetic	Simple insertion
Boolean	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion
Numeric edited	All

Figure 4-7. Valid Editing for Each Data Category

Category	Valid Insertion Symbols
Alphabetic	B
Boolean	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	B 0 /
Numeric edited	B 0 / ,

Figure 4-8. Valid Insertion Symbols for Simple Insertion Editing for Each Data Category

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

Figure 4-9. Examples of Simple Insertion Editing

Special Insertion Editing: This type of editing is valid only for numeric edited items.

The period is the special insertion symbol; it also represents the actual decimal point for alignment purposes.

The period insertion symbol is counted in the size of the item, and represents the position within the item where the actual decimal point will be inserted.

The actual decimal point and the assumed decimal point (the symbol V) must not both be specified in one PICTURE character-string.

Examples of special insertion editing are shown in Figure 4-10.

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

'cs' (currency symbol)

+ - CR DB (editing sign control symbols).

- In fixed insertion editing, only one currency symbol and one editing sign control symbol can be specified in one PICTURE character-string.
- Unless it is preceded by a + or - symbol, the currency symbol must be the leftmost character position in the character-string.
- When either + or - is used as a symbol, it must represent either the leftmost or rightmost character position in the character-string.

PICTURE	Value of Data	Edited Results
999.99	1.234	001.23
999.99	12 34	012.34
999.99	123.45	123.45
999.99	1234.5	234.50

Figure 4-10. Examples of Special Insertion Editing

- When CR or DB is used as a symbol, it must represent the rightmost two character positions in the character-string.

- Editing sign control symbols produce results depending on the value of the data item as shown in Figure 4-11.

Examples of fixed insertion editing are shown in Figure 4-12.

Editing Symbol in PICTURE Character String	Resulting Data Item Positive or Zero	Resulting Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Figure 4-11. Editing Sign Control Results

PICTURE	Value of Data	Edited Result
999.99+	+6555.556	555.55+
+9999.99	-6555.555	-6555.55
9999.99	+1234.56	1234.56
\$999.99	-123.45	\$123.45
-\$999.99	-123.456	-\$123.45
\$9999.99CR	+123.45	\$0123.45
\$9999.99DB	-123.45	\$0123.45DB

Figure 4-12. Examples of Fixed Insertion Editing

Floating Insertion Editing: This type of editing is valid only for numeric edited items. The following symbols are used:

'cs' + -

Within one PICTURE character-string, these symbols are mutually exclusive as floating insertion characters.

Floating insertion editing is specified by using a string of at least two of the allowable floating insertion symbols to represent leftmost character positions in which these characters can be inserted.

The leftmost floating insertion symbol in the character-string represents the leftmost limit at which this character can appear in the data item. The rightmost floating insertion symbol represents the rightmost limit at which this character can appear.

The second leftmost floating insertion symbol in the character-string represents the leftmost limit at which numeric data can appear within the data item. Nonzero numeric data can replace all characters at or to the right of this limit.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating insertion symbols are considered part of the floating character-string.

In a PICTURE character-string there are two methods to represent floating insertion editing and to perform editing:

- Any or all leading numeric character positions to the left of the decimal point are represented by the floating insertion symbol. When editing is performed, a single floating insertion character is placed to the immediate left of the first nonzero digit in the data or of the decimal point, whichever is farther left. The character positions to the left of the inserted character are filled with spaces.
- All the numeric character positions are represented by the floating insertion symbol. When editing is performed, then:
 - If the value of the data is zero, the entire data item will contain spaces.
 - If the value of the data is nonzero, the result is the same as in method 1.

To avoid truncation, the minimum size of the PICTURE character-string must be the sum of:

- The number of character positions in the sending item
- The number of nonfloating insertion symbols in the receiving item
- One character for the floating insertion symbol.

Examples of floating insertion editing are shown in Figure 4-13.

Zero Suppression and Replacement Editing: This type of editing is valid only for numeric edited items.

The symbols Z and * are used for zero suppression. These symbols are mutually exclusive in the PICTURE clause.

The following symbols are mutually exclusive as floating replacement symbols in one PICTURE character-string:

Z * + - 'cs'

Zero suppression editing is specified by using a string of one or more of the allowable symbols to represent leftmost character positions in which zero suppression and replacement editing can be performed.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating editing symbols are considered part of the string.

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

Figure 4-13. Examples of Floating Insertion Editing

In a PICTURE character-string, there are two ways to represent zero suppression and perform editing:

- Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols. When editing is performed, any leading zero in the data that appears in the same character position as a suppression symbol is replaced by the replacement character. Suppression stops at the character farthest left that:
 - Does not correspond to a suppression symbol.
 - Contains nonzero data.
 - Is the decimal point.
- All the numeric character positions in the PICTURE character-string are represented by the suppression symbols. When editing is performed and the value of the data is nonzero, the result is the same as in the preceding rule. The following rules apply if the value of the data is zero:
 - If Z has been specified, the entire data item contains spaces.
 - If * has been specified, the entire data item, except the actual decimal point, contains asterisks.

The asterisk as a suppression symbol and the BLANK WHEN ZERO clause must not be specified for the same entry.

Examples of zero suppression and replacement editing are shown in Figure 4-14.

PICTURE	Value of Data	Edited Result
****. **	0000.00	****. **
ZZZZ.ZZ	0000.00	
ZZZZ.99	0000.00	.00
****.99	0000.00	****.00

Figure 4-14 (Part 1 of 2). Examples of Zero Suppression and Replacement Editing

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

Figure 4-14 (Part 2 of 2). Examples of Zero Suppression and Replacement Editing

RENAMES Clause

The RENAMES clause specifies alternative, possibly overlapping, groupings of elementary data items. This clause allows a single data-item to rename a group of data items within a record.

Format

$$66 \text{ data-name-1 } \underline{\text{RENAMES}} \text{ data-name-2 } \left[\left\{ \begin{array}{c} \underline{\text{THROUGH}} \\ \underline{\text{THRU}} \end{array} \right\} \text{ data-name-3} \right].$$

Note: Level-number 66 and data-name-1 are not part of the RENAMES clause itself, and are included in the format only for clarity.

One or more RENAMES entries can be written for a logical record. All RENAMES entries associated with one logical record must immediately follow that record's last data description entry.

Data-name-1 identifies an alternative grouping of data items. It cannot be used as a qualifier; it can be qualified only by the names of FD or SD entries or level-01 entries.

A level-66 entry cannot rename a level-01, level-77, level-88, or another level-66 entry.

Data-name-2 or data-name-3 identifies the original grouping of elementary data items; that is, they must name elementary or group items within the associated level-01 entry and must not be the same data-name. Both data-names may be qualified.

The OCCURS clause must not be specified in the data entries for data-name-2 and data-name-3, or for any group entry to which they are subordinate. In addition, the OCCURS DEPENDING ON clause must not be specified for any item occupying storage between data-name-2 and data-name-3.

When data-name-2 is specified, and data-name-3 is not specified, data-name-1 is defined with the same attributes as data-name-2.

When both data-name-2 and data-name-3 are specified, the following occurs:

- If data-name-2 is an elementary item, data-name-1 is defined as a group item starting with data-name-2 and ending with data-name-3, or the last elementary item in data-name-3, if it is a group item.
- If data-name-2 is a group item, data-name-1 is defined as a group item starting with the first elementary item in data-name-2, and ending with data-name-3, or the last elementary item in data-name-3, if it is a group item.

The keywords THRU and THROUGH are equivalent.

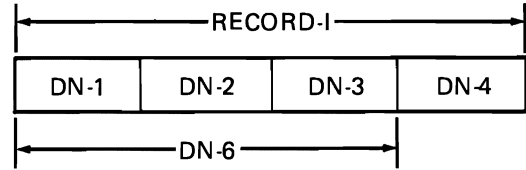
The leftmost character in data-name-3 must not precede that in data-name-2; the rightmost character in data-name-3 must follow that in data-name-2. This means that data-name-3 cannot be subordinate to data-name-2.

Valid and invalid specifications of the RENAMES clause are given in Figure 4-15.

Example 1 (Valid)

```

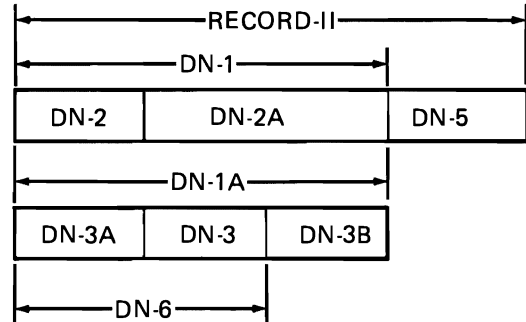
01 RECORD-I.
05 DN-1 . . . .
05 DN-2 . . . .
05 DN-3 . . . .
05 DN-4 . . . .
66 DN-6 RENAMES DN-1 THROUGH DN-3.
    
```



Example 2 (Valid)

```

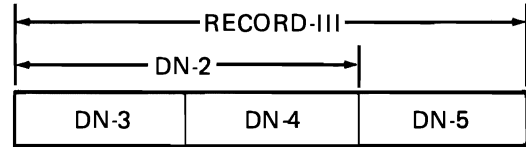
01 RECORD-II.
05 DN-1.
10 DN-2 . . . .
10 DN-2A . . . .
05 DN-1A REDEFINES DN-1.
10 DN-3A . . . .
10 DN-3 . . . .
10 DN-3B . . . .
05 DN-5 . . . .
66 DN-6 RENAMES DN-2 THROUGH DN-3.
    
```



Example 3 (Invalid)

```

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

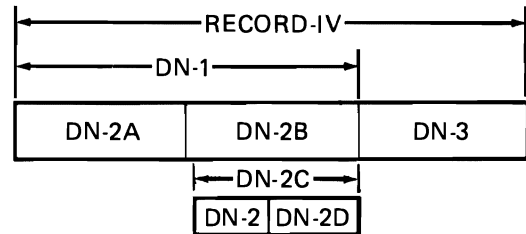


DN-6 is indeterminate

Example 4 (Invalid)

```

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



DN-4 is indeterminate

Figure 4-15. Valid and Invalid Specifications of the RENAMES Clause



Chapter 5. Procedure Division

Procedure Division Concepts

The Procedure Division is required in every COBOL source program. The Procedure Division consists of optional Declaratives and procedures that contain the sections and/or paragraphs, sentences, and statements that solve a data processing problem.

Declaratives

The Declarative section provides a method of invoking procedures that are executed when an exceptional condition occurs that is to be tested by the COBOL programmer.

When Declarative sections are specified, they must be grouped at the beginning of the Procedure Division. Declarative sections are preceded by the keyword `DECLARATIVES` and followed by the keywords `END DECLARATIVES`.

If Declarative sections are specified, the entire Procedure Division must be divided into sections.

Procedures

A *procedure* is a paragraph, group of paragraphs, a section, or a group of sections within the Procedure Division. A *procedure-name* is a user-defined name that identifies a section or a paragraph.

A *section* consists of a section header followed by zero, one, or more than one successive paragraphs. A *section-header* is a section-name followed by the keyword `SECTION`, an optional segment-number, followed by a period and a space. Segment-numbers are explained under “Segmentation Feature” in Chapter 6. A *section-name* is a user-defined word that identifies a section. A section-name, because it cannot be qualified, must be unique. A section ends immediately before the next section header, at the end of the Procedure Division, or, in the Declaratives portion, at the keywords `END DECLARATIVES`.

A *paragraph* consists of a paragraph-name followed by a period and a space. Zero, one, or more than one successive sentences are allowed. A *paragraph-name* is a user-defined word that identifies a paragraph. A paragraph-name, because it can be qualified, need not be unique. A paragraph ends immediately before the next paragraph-name or section header, at the end of the Procedure Division, or, in the Declaratives portion, at the keywords END DECLARATIVES. If one paragraph in a program is contained within a section, then all paragraphs must be contained in sections.

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

A *statement* is a syntactically valid combination of words (identifiers, data-names, figurative constants, and so on) and symbols (literals, relational-operators, and so on) beginning with a COBOL verb.

A *data-name* is a user-defined word naming a data item described in a data description entry in the Data Division. When *data-name* is used in a general format, it represents a word that cannot be subscripted, indexed, or qualified unless this is specifically permitted by the rules for that format.

An *identifier* consists of the word or words necessary to make unique reference to a data item through qualification, subscripting, or indexing. In any Procedure Division reference except the class test, if the contents of an identifier are not compatible with the class specified through its PICTURE clause, results are unpredictable.

Note: A level-88 (condition-name) entry, because it is not a data item, cannot be an identifier. The associated conditional variable, however, can be an identifier.

Procedure Division Organization

The structure of the Procedure Division is shown in the following formats. Figure 5-1 gives an example.

Format 1

```

PROCEDURE DIVISION [ USING data-name-1 [, data-name-2] . . . ].
[ DECLARATIVES.
{section-name SECTION [segment-number] . use-sentence.
[paragraph-name. [sentence] . . . } . . . } . . .
END DECLARATIVES.]
{section-name SECTION [segment-number] .
[paragraph-name. [sentence] . . . } . . . } . . .

```

Format 2

```

PROCEDURE DIVISION [ USING data-name-1 [, data-name-2] . . . ].
{paragraph-name. [sentence] . . . } . . .

```

Coding Example

SEQUENCE	(PAGE)	SERIAL	CO	A	B
1	3	4	6	7	8
004	0	1	0	PROCEDURE DIVISION.	
	0	2	0	DECLARATIVES.	
	0	3	0	SECTION-NAME SECTION.	
	0	4	0	PARAGRAPH-NAMES.	
	0	5	0	PROGRAMMING STATEMENTS.	
	0	6	0	COMMENTS.	
	0	7	0	END DECLARATIVES.	
	0	8	0	SECTION-NAME SECTION.	
	0	9	0	PARAGRAPH-NAME.	
	1	0	0	PROGRAMMING STATEMENTS.	
	1	1			

Figure 5-1. Coding Example to Show Procedure Division Organization

Categories of Sentences

There are three categories of sentences: conditional sentences, imperative sentences, and compiler-directing sentences.

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

An *imperative sentence* is an imperative statement, which can consist of a series of imperative statements, followed by a period and a space.

A *compiler-directing sentence* is a single compiler-directing statement, followed by a period and a space.

Categories of Statements

Three categories of statements are used in COBOL: conditional statements, imperative statements, and compiler-directing statements.

A *conditional statement* specifies that the truth value of a condition is to be determined, and that the subsequent action of the object program is dependent on this truth value. Figure 5-2 lists types of COBOL conditional statements.

An *imperative statement* specifies that an unconditional action is to be taken by the object program. An imperative statement can also consist of a series of imperative statements. Figure 5-3 lists types of COBOL imperative statements.

A *compiler-directing statement* causes the compiler to take a specific action during compilation. Figure 5-4 lists types of COBOL compiler-directing statements.

Type	Conditional Statement
Decision	IF
Input/Output	DELETE... INVALID KEY READ... AT END READ... INVALID KEY REWRITE... INVALID KEY START... INVALID KEY WRITE... AT END-OF-PAGE WRITE... INVALID KEY
Arithmetic	ADD... ON SIZE ERROR COMPUTE... ON SIZE ERROR DIVIDE... ON SIZE ERROR MULTIPLY... ON SIZE ERROR SUBTRACT... ON SIZE ERROR
Data Movement	STRING... ON OVERFLOW UNSTRING... ON OVERFLOW
Table Handling	SEARCH
Ordering	RETURN... AT END
Inter-program Communication	CALL... ON OVERFLOW
Procedure Branching	PERFORM... UNTIL

Figure 5-2. Types of Conditional Statements

-
- 1 Without the SIZE ERROR phrase.
 - 2 Without the INVALID KEY phrase.
 - 3 Without the ON OVERFLOW phrase.
 - 4 Without the AT END, INVALID KEY, or NO DATA phrase.

Type	Imperative Statement
Arithmetic	ADD ¹ COMPUTE ¹ DIVIDE ¹ INSPECT (TALLYING) MULTIPLY ¹ SUBTRACT ¹
Data Movement	ACCEPT (DATE, DAY, TIME) INSPECT (REPLACING) MOVE STRING ³ UNSTRING ³
Ending	STOP RUN EXIT PROGRAM
Input/Output	ACCEPT OPEN ACQUIRE READ ⁴ CLOSE REWRITE ² COMMIT ROLLBACK DELETE ² START ² DISPLAY STOP literal DROP WRITE ⁵
Ordering	MERGE RELEASE SORT
Procedure Branching	ALTER EXIT GO PERFORM ⁶
Table Handling	SET
Inter-program Communication	CALL ³ CANCEL

Figure 5-3. Types of Imperative Statements

Type	Compiler-Directing Statement
Library	COPY
Declarative	USE
Documentation	ENTER

Figure 5-4. Types of Compiler-Directing Statements

Categories of Expressions

Two categories of expressions are used in COBOL: arithmetic expressions and conditional expressions.

Arithmetic expressions are used as operands of conditional or arithmetic statements.

Conditional expressions cause the object program to select alternative paths of control, depending on the value of a truth test. There are two types

⁵ Without the INVALID KEY or END-OF-PAGE phrase.

⁶ Without the UNTIL phrase.

of conditional expressions: simple conditions and complex conditions. Conditional expressions can be specified in the IF, PERFORM, and SEARCH statements.

Sample Procedure Division Statements

```
. 1 . . . . . 2 . . . . . 3 . . . . . 4 . . . . . 5 . . . . . 6 . . . . . 7
PROCEDURE DIVISION.
DECLARATIVES.
ERROR-IT SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON INPUT-DATA.
ERROR-ROUTINE.
    IF CHECK-IT = "30" ADD 1 TO DECLARATIVE-ERRORS.
END DECLARATIVES.
BEGIN-NON-DECLARATIVES SECTION.
100-BEGIN-IT.
    OPEN INPUT INPUT-DATA OUTPUT REPORT-OUT.
110-READ-IT.
    READ INPUT-DATA RECORD
        AT END MOVE "Y" TO EOF-SW.
    IF EOF-SW NOT = "Y" ADD 1 TO RECORDS-IN.
200-MAIN-ROUTINE.
    PERFORM PROCESS-DATA UNTIL EOF-SW = "Y".
    PERFORM FINAL-REPORT THRU FINAL-REPORT-EXIT.
    DISPLAY "TOTAL RECORDS IN = " RECORDS-IN UPON WORK-STATION.
    DISPLAY "DECLARATIVE ERRORS = " DECLARATIVE-ERRORS
        UPON WORK-STATION.
    STOP RUN.
PROCESS-DATA.
    IF RECORD-ID = "G"
        PERFORM PROCESS-GEN-INFO
    ELSE
        IF RECORD-CODE = "C"
            PERFORM PROCESS-SALES-DATA
        ELSE
            PERFORM UNKNOWN-RECORD-TYPE.
```

Arithmetic Expressions

Arithmetic expressions are used as operands of certain conditional and arithmetic statements. An arithmetic expression can consist of any of the following:

- An identifier described as a numeric elementary item
- A numeric literal
- Identifiers and literals, as defined in Items 1 and 2, separated by arithmetic operators
- Two arithmetic expressions, as defined in Items 1, 2, and/or 3, separated by an arithmetic operator
- An arithmetic expression, as defined in Items 1, 2, 3, and/or 4, enclosed in parentheses.

Any arithmetic expression can be preceded by a unary operator.

Identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic can be performed.

Arithmetic Operators

The five binary arithmetic operators and two unary arithmetic operators shown in Figure 5-5 can be used in arithmetic expressions. The arithmetic operators are represented by specific characters that must be preceded and followed by a space.

Parentheses can be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first. When expressions are contained within a nest of parentheses, evaluation proceeds from the innermost to the outermost set of parenthetical expressions.

When parentheses are not used, or when parenthesized expressions are at the same level of inclusiveness, the following hierarchical order is implied:

1. Unary operator
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction.

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

Unary Operator	Meaning
+	Multiplication by +1; retains original sign
-	Multiplication by -1; changes sign

Figure 5-5. Binary and Unary Operators

Parentheses either eliminate ambiguities in logic where consecutive operations appear at the same hierarchical level or modify the normal hierarchical sequence of execution when this is necessary. When the order of consecutive operations at the same hierarchical level is not completely specified by parentheses, the order is from left to right.

Figure 5-6 shows permissible arithmetic symbol pairs. An arithmetic symbol pair is the appearance of two such symbols in sequence.

An arithmetic expression can begin only with a left parenthesis, a unary operator, or a variable (that is, an identifier or literal). An arithmetic expression can end only with a right parenthesis or a variable. An arithmetic expression must contain at least one reference to an identifier or literal. There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression; each left parenthesis is placed to the left of its corresponding right parenthesis.

Programming Notes: The results of exponentiation are truncated after the thirteenth fractional digit. The results of exponentiation when the exponent is noninteger are accurate to seven digits.

First Symbol	Second Symbol				
	Variable (identifier or literal)	* / ** + -	Unary + Unary -	()
Variable (identifier or literal)	-	p	-	-	p
* / ** + -	p	-	p	p	-
Unary + or Unary -	p	-	-	p	-
(p	-	p	p	-
)	-	p	-	-	p
<p>p indicates a permissible pairing - indicates that the pairing is not permitted</p>					

Figure 5-6. Valid Arithmetic Symbol Pairs

Conditional Expressions

A conditional expression causes the object program to select alternative paths of control, depending on the truth value of a test. Conditional expressions can be specified in IF, PERFORM, and SEARCH statements. A conditional expression can be specified in simple conditions and in complex conditions. Both simple and complex conditions can be enclosed within any number of paired parentheses; parentheses do not change the category of the condition.

Simple Conditions

There are five simple conditions: class condition, condition-name condition, relation condition, sign condition, and switch-status condition. A simple condition has a truth value of true or false. When a simple condition is enclosed in paired parentheses, its truth value is not changed.

Class Condition

The class condition determines whether a data item is alphabetic or numeric.

Format

$$\text{identifier IS [NOT] } \left\{ \begin{array}{l} \text{NUMERIC} \\ \text{ALPHABETIC} \end{array} \right\}$$

The identifier being tested must be described implicitly or explicitly as USAGE DISPLAY. This identifier is determined to be numeric only if the contents consist of any combination of the digits 0 through 9.

If the PICTURE of the identifier being tested does not contain an operational sign, the identifier is determined to be numeric only if the contents are numeric and an operational sign is not present.

If the PICTURE of the identifier being tested does contain an operational sign, the identifier is determined to be numeric only if the item is an elementary item, the contents are numeric, and a valid operational sign is present.

In the EBCDIC collating sequence, valid embedded operational positive signs are hexadecimal F, C, E, and A. Negative signs are hexadecimal D and B. The preferred positive sign is hexadecimal F, and the preferred negative sign is hexadecimal D. For items described with the SIGN IS SEPARATE clause, valid operational signs are + (hex 4E) and - (hex 60).

The NUMERIC test cannot be used with an identifier described either as alphabetic or as a group item that contains one or more signed elementary items. The identifier being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters A through Z and the space.

The ALPHABETIC test cannot be used with an identifier described as numeric.

Figure 5-7 shows valid forms of the class test.

Type of Identifier	Valid Forms of the Class Test
Alphabetic	ALPHABETIC NOT ALPHABETIC
Alphanumeric, alphanumeric edited, or numeric edited	ALPHABETIC NOT ALPHABETIC NUMERIC NOT NUMERIC
Zoned decimal	NUMERIC NOT NUMERIC

Figure 5-7. Valid Forms of the Class Test

Condition-Name Condition

A condition-name condition causes a conditional variable to be tested to determine whether its value is equal to any of the values associated with the condition-name (level-88 item).

Format

condition-name

A condition-name is used in conditions as an abbreviation for the relation condition, because the specified condition-name is equal to only one of the values or ranges of values assigned to the specified conditional variable. The result of the test is true if one of the values corresponding to the condition-name equals the current value of the associated conditional variable.

If the condition-name is associated with a range of values or with several ranges of values, the conditional variable is tested to determine whether or not its value falls within the range(s), including the end values. The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

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

```

02 GRADE-ID PIC 99.
   88 PRIMARY-OTHER
      VALUE 1 THRU 3, 5, 6.
   88 PRIMARY-FOUR
      VALUE 4.
   88 JUNIOR-HI
      VALUE 7 THRU 9.
   88 SENIOR-HI
      VALUE 10 THRU 12.

```

GRADE-ID is the conditional variable, PRIMARY-OTHER, PRIMARY-FOUR, JUNIOR-HI, and SENIOR-HI are condition-names. For individual records in the file, only one of the values specified in the condition-name entries can be present. To determine the grade level of a specific record, any of the following can be coded:

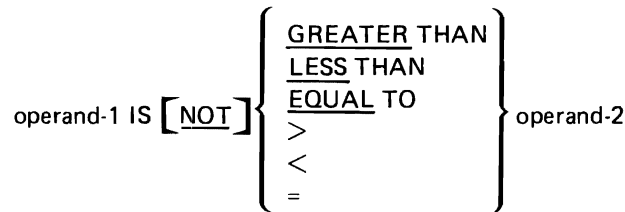
IF PRIMARY-OTHER...
 (which tests for values 1, 2, 3, 5, 6)
 IF PRIMARY-FOUR...
 (which tests for value 4)
 IF JUNIOR-HI...
 (which tests for values 7, 8, 9)
 IF SENIOR-HI...
 (which tests for values 10, 11, 12)

Depending on the evaluation of the condition-name condition, alternative paths of execution are taken by the object program.

Relation Condition

A relation condition causes a comparison between two operands, either of which can be an identifier, a literal, or an arithmetic expression.

Format



Operand-1 is the subject of the relation condition; operand-2 is the object of the relation condition. Operand-1 and operand-2 can each be an identifier, a literal, or an arithmetic expression. The relation condition must contain at least one reference to an identifier. Except when two numeric operands are compared, operand-1 and operand-2 must have the same USAGE.

The relational operator specifies the type of comparison to be made. Figure 5-8 shows relational operators and their meanings. Each relational operator must be preceded and followed by a space.

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

Figure 5-8. Relational Operators and Their Meanings

Rules for numeric and nonnumeric comparisons are given in the following paragraphs. If either of the operands is a group item, nonnumeric comparison rules apply. Figure 5-9 summarizes the permissible comparisons.

First Operand	Second Operand														
	GR	AL	AN	ANE	NE	FC NNL	ZR NL	ZD	BI	PD	AE	BO	IN	IDI	
Group (GR)	NN	NN	NN	NN	NN	NN	NN	NN							
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN	NN							
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN	NN							
Alphanumeric edited (ANE)	NN	NN	NN	NN	NN	NN	NN	NN							
Numeric edited (NE)	NN	NN	NN	NN	NN	NN	NN	NN							
Figurative constants, except ZERO (FC) and nonnumeric literal (NNL)	NN	NN	NN	NN	NN			NN							
Figurative constant ZERO (ZR) and numeric literal (NL)	NN	NN	NN	NN	NN			NU	NU	NU	NU		IO ¹		
Zoned decimal (ZD)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU		IO ¹		
Binary (BI)							NU	NU	NU	NU	NU		IO ¹		
Packed decimal (PD)							NU	NU	NU	NU	NU		IO ¹		
Arithmetic expression (AE)							NU	NU	NU	NU	NU				
Boolean data item (BO) or Boolean literal												BO			
Index-name (IN)							IO ¹	IO ¹	IO ¹	IO ¹			IO	IV	
Index data item (IDI)													IV	IV	

BO = Comparison as described for Boolean operands.
 NN = Comparison as described for nonnumeric operands.
 NU = Comparison as described for numeric operands.
 IO = Comparison as described for two index-names (by occurrence number).
 IV = Comparison as described for index data items (by value).
 Blank = Comparison is not allowed.

¹Valid only if the numeric item is an integer.

Figure 5-9. Permissible Comparisons of Operands

Comparison of Numeric Operands: For numeric class operands, algebraic values are compared. The length (number of digits) of the operands is not significant. Zero is considered a unique value, regardless of the sign. Unsigned numeric operands are considered positive; regardless of their USAGE, comparison of numeric operands is permitted.

IBM Extension

Comparison of Boolean Operands: Boolean operands can be used only in the [NOT] EQUAL TO relation condition. Boolean operands cannot be compared to non-Boolean operands. Boolean data items and literals must be one position in length. Two Boolean operands are equal if they both have a value of Boolean 1 or Boolean 0. The Boolean operands are unequal if one has a value of Boolean 1 and the other has a value of Boolean 0.

End of IBM Extension

Comparison of Nonnumeric Operands: A comparison of two nonnumeric operands or of one numeric and one nonnumeric operand is made with respect to the program collating sequence in use.

When a nonnumeric and a numeric operand are compared, the following rules apply:

- If the nonnumeric operand is a literal or an elementary data item, the numeric operand is treated as though it were moved to an alphanumeric elementary data item of the same size, and the contents of this alphanumeric data item were then compared with the nonnumeric operand.
- If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size, and the contents of this group item were then compared with the nonnumeric operand. For further discussion of the rules for alphanumeric and group move operations, see the “MOVE Statement” later in this chapter.

Numeric and nonnumeric operands can be compared only when their USAGE, explicitly or implicitly, is the same. In such comparisons, the numeric operand must be described as an integer literal or data item; noninteger literals and data items must not be compared with nonnumeric operands.

The size of each operand is the total number of characters in that operand; the size affects the result of the comparison. There are two kinds of operands to consider: operands of equal size and operands of unequal size.

Operands of equal size: Characters in corresponding positions of the two operands are compared, beginning with the leftmost character and continuing through the rightmost character.

If all pairs of characters through the last pair test as equal, the operands are considered equal. If a pair of unequal characters is encountered, the characters are tested to determine their relative positions in the collating

sequence. The operand containing the character higher in the sequence is considered the greater operand.

Operands of unequal size: If the operands are of unequal size, the comparison is made as though the shorter operand were extended to the right with enough spaces to make the operands equal in size.

Note: Valid comparisons for index-names and index data items are discussed under “Table Handling” in Chapter 6.

Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand is less than, greater than, or equal to zero.

Format

$$\text{operand IS } [\text{NOT}] \left\{ \begin{array}{l} \text{POSITIVE} \\ \text{NEGATIVE} \\ \text{ZERO} \end{array} \right\}$$

The operand being tested must be defined as a numeric identifier or as an arithmetic expression that contains at least one reference to an identifier.

The operand is POSITIVE if its value is greater than zero, NEGATIVE if its value is less than zero, and ZERO if its value is equal to zero. An unsigned operand is POSITIVE or ZERO.

When NOT is specified, one algebraic test is executed for the truth value of the sign condition. For example, NOT ZERO is regarded as true when the operand tested is positive or negative in value.

Switch-Status Condition

The switch-status condition determines the on or off status of an UPSI switch.

Format

condition-name

The condition-name must be defined to be associated with the ON or OFF value of a switch in the SPECIAL-NAMES paragraph.

The switch-status condition tests the value associated with the condition-name. The result of the test is true if the UPSI switch is set to the position corresponding to condition-name.

Complex Conditions

A complex condition is a condition in which one or more logical operators act upon one or more conditions. Complex conditions include:

- Negated simple conditions
- Combined conditions
- Negated combined conditions.

Each logical operator must be preceded and followed by a space. The logical operators and their meanings are shown in Figure 5-10.

The truth value of a complex condition depends on the truth values of the simple conditions and negated simple conditions that make up the complex condition. The logical operators tell the compiler how to combine these individual truth values.

Logical Operator	Meaning
AND	Logical conjunction – the truth value is true when both conditions are true.
OR	Logical inclusive OR – the truth value is true when either or both conditions are true.
NOT	Logical negation – reversal of truth value (the truth value is true if the condition is false).

Figure 5-10. Logical Operators and Their Meanings

Negated Simple Conditions

A simple condition is negated through the use of the logical operator NOT.

Format

NOT simple-condition

The simple-condition to be negated can be a class condition, a condition-name condition, a relation condition, a sign condition, or a switch-status condition. The simple-condition cannot be negated if the condition itself contains a NOT.

The negated simple-condition gives the opposite truth value as the simple condition. That is, if the truth value of the simple-condition is true, then the truth value of that same negated simple-condition is false.

Placing a negated simple-condition within parentheses does not change its truth value. For example, the following two statements are equivalent:

NOT A IS EQUAL TO B

NOT (A IS EQUAL TO B)

Combined Conditions

Two or more conditions can be logically connected to form a combined condition.

Format

$$\text{condition} \left\{ \left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\} \text{condition} \right\} \dots$$

The condition to be combined can be a simple-condition, a negated simple-condition, a combined condition, a negated combined condition (that is, the NOT logical operator followed by a combined condition enclosed in parentheses). Combinations of the preceding conditions are specified according to the rules given in Figure 5-11.

Parentheses are never needed when either AND or OR (but not both) are used exclusively in one combined condition. However, parentheses might be needed to find a final truth value when a combination of AND, OR, and NOT is used. There must be a one-to-one correspondence between left and right parentheses with each left parenthesis to the left of its corresponding right parenthesis.

Figure 5-11 summarizes the way in which conditions and logical operators can be combined and parenthesized. Figure 5-12 illustrates the relationships between logical operators and conditions C1 and C2 where C1 and C2 are any conditions as defined above.

Evaluating Conditional Expressions: If parentheses are used, logical evaluation of combined conditions proceeds in the following order:

1. Conditions within parentheses are evaluated first.
2. Within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition.

If parentheses are not used (or are not at the same level of inclusiveness), the combined condition is evaluated in the following order:

1. Arithmetic expressions.

Condition Element	Permissible Position in Conditional Expressions			
	Leftmost	When Not Leftmost, Can Be Immediately Preceded By:	When Not Rightmost, Can Be Immediately Followed By:	Rightmost
Simple-Condition	Yes	OR NOT AND (OR AND)	Yes
OR AND	No	Simple-Condition)	Simple-Condition NOT (No
NOT	Yes	OR AND (Simple-Condition (No
(Yes	OR NOT AND (Simple-Condition NOT (No
)	No	Simple-Condition)	OR AND)	Yes

Figure 5-11. Valid Combinations of Conditions, Logical Operators, and Parentheses in a Conditional Expression

Values for C1	Values for C2	C1 AND C2	C1 OR C2	NOT (C1 AND C2)	NOT C1 AND C2	NOT (C1 OR C2)	NOT C1 OR C2
True	True	True	True	False	False	False	True
False	True	False	True	True	True	False	True
True	False	False	True	True	False	False	False
False	False	False	False	True	False	True	True

Figure 5-12. How Logical Operators Affect the Evaluation of Conditions

2. Simple-conditions in the following order:
 - a. Relation
 - b. Class
 - c. Condition-name
 - d. Switch-status
 - e. Sign.
3. Negated simple-conditions in the same order as item 2.
4. Combined conditions, in the following order:
 - a. AND
 - b. OR.
5. Negated combined conditions in the following order:
 - a. AND
 - b. OR.
6. Consecutive operands at the same evaluation-order level are evaluated from left to right. However, the truth value of a combined condition can sometimes be determined without evaluating the truth value of all the component conditions.

The component conditions of a combined condition are evaluated from left to right. If the truth value of one condition is not affected by the evaluation of further elements of the combined condition, these elements are not evaluated. However, the truth value of the condition will always be the same (as if the condition had been evaluated in full), as described earlier in this paragraph.

For example:

```
NOT A IS GREATER THAN B OR A + B IS EQUAL  
TO C AND D IS POSITIVE
```

is evaluated as if parenthesized as follows:

```
(NOT (A IS GREATER THAN B)) OR (((A+B) IS EQUAL  
TO C) AND (D IS POSITIVE))
```

The order of evaluation in this example is as follows:

1. (NOT (A IS GREATER THAN B)) is evaluated. If true, the rest of the condition is not evaluated, as the expression is true.
2. (A+B) is evaluated, giving some intermediate result, x.
3. (x IS EQUAL TO C) is evaluated. If false, the rest of the condition is not evaluated, as the expression is false.
4. (D IS POSITIVE) is evaluated, giving the final truth value of the expression.

Abbreviated Combined Relation Conditions

When relation-conditions are written consecutively and no parentheses are used within the consecutive sequence, any relation-condition after the first can be abbreviated by either:

- Omission of the subject
- Omission of the subject and relation operator.

Format

$$\text{relation-condition} \left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\} \text{ [NOT] } \left[\begin{array}{l} \text{GREATER THAN} \\ \text{LESS THAN} \\ \text{EQUAL TO} \\ > \\ < \\ = \end{array} \right] \text{ object} \dots$$

In any consecutive sequence of relation-conditions, both forms of abbreviation can be specified. The abbreviated condition is evaluated as if:

- The last stated subject is the missing subject.
- The last stated relational operator is the missing relational operator.
- The resulting combined condition must comply with the rules for element sequence in combined conditions, as shown in Figure 5-11.
- The word NOT is considered part of the relational operator in the forms NOT GREATER THAN, NOT >, NOT LESS THAN, NOT <, NOT EQUAL TO, and NOT =.
- NOT in any other position is considered a logical operator, and thus results in a negated related-condition.

Figure 5-13 shows examples of abbreviated combined relation-conditions and their nonabbreviated equivalents.

Abbreviated Combined Relation-Condition	Nonabbreviated Equivalent
A = B AND NOT LESS THAN C OR D	((A = B) AND (A NOT LESS THAN C)) OR (A NOT LESS THAN D)
A NOT GREATER THAN B OR C	(A NOT GREATER THAN B) OR (A NOT GREATER THAN C)
NOT A = B OR C	(NOT (A = B) OR (A = C))
NOT (A = B OR LESS THAN C)	NOT ((A = B) OR (A LESS THAN C))

Figure 5-13 (Part 1 of 2). Abbreviated Combined Relation-Condition Equivalent

Abbreviated Combined Relation-Condition	Nonabbreviated Equivalent
NOT (A NOT = B AND C AND NOT D)	NOT (((A NOT = B) AND (A NOT = C)) AND (NOT (A NOT = D)))

Figure 5-13 (Part 2 of 2). Abbreviated Combined Relation-Condition Equivalent

Declaratives

The Declaratives section provides a method of invoking procedures that are executed when an exceptional condition occurs that cannot normally be tested by the COBOL programmer. Declarative procedures are provided for the processing of exceptional input/output conditions and debugging procedures.

Format

```

PROCEDURE DIVISION [ USING data-name-1 [ , data-name-2 ] . . . ].
[ DECLARATIVES.
{ section-name SECTION [ segment-number ] . USE statement.
[ paragraph-name. [ sentence ] . . . } . . .
END DECLARATIVES.]

```

Declarative procedures are written at the beginning of the Procedure Division in a series of Declarative sections. Each such section is preceded by a USE statement that identifies under what conditions the section is used. The series of procedures that follow specify what actions are to be taken when the exceptional condition occurs. Each Declarative section ends with the occurrence of another section-name followed by a USE statement, or with the keywords END DECLARATIVES.

The entire group of Declarative procedures is preceded by the keyword DECLARATIVES, written on the next line after the Procedure Division header; the group is followed by the keywords END DECLARATIVES. The keywords DECLARATIVES and END DECLARATIVES must each begin in Area A and be followed by a period. No other text can appear on the same line.

In the Declaratives portion of the Procedure Division, each section header (with an optional segment number) must be followed by a period and a space, and must be followed by a USE statement followed by a period and a space. No other text can appear on the same line. There are two forms of the USE statement:

- USE AFTER EXCEPTION/ERROR
- USE FOR DEBUGGING.

The USE statement itself is never executed; instead, the USE statement defines the conditions that will cause execution of the immediately following procedural paragraphs, which specify the actions to be taken. After the procedure is executed, control is returned to the routine that activated it.

Within a Declarative procedure, except for the USE statement itself, there must be no reference to any nondeclarative procedure.

Within a Declarative procedure, no statement can be executed that would cause execution of a USE procedure that has been previously invoked and has not yet returned control to the invoking routine.

An exit from a Declarative procedure is effected by executing the last statement in the procedure.

In this chapter, only the USE AFTER EXCEPTION/ERROR Declaratives procedure is described. The USE FOR DEBUGGING Declaratives procedure is described under “Debugging Features” in Chapter 6.

EXCEPTION/ERROR Declarative

The EXCEPTION/ERROR Declarative specifies procedures for input/output exception or error handling that are to be executed in addition to the standard system procedures.

Format

$$\text{USE AFTER STANDARD } \left\{ \begin{array}{l} \text{EXCEPTION} \\ \text{ERROR} \end{array} \right\} \text{ PROCEDURE ON } \left\{ \begin{array}{l} \text{file-name-1 } [, \text{file-name-2}] \dots \\ \text{INPUT} \\ \text{OUTPUT} \\ \text{I-O} \\ \text{EXTEND} \end{array} \right\} .$$

The words EXCEPTION and ERROR are synonymous and can be used interchangeably.

File-Name Phrase

This phrase is valid for sequential, indexed, and relative files. When this phrase is specified, the procedure is executed only for the file(s) named. No file-name can refer to a sort-merge file. For any given file, only one EXCEPTION/ERROR procedure can be specified. For example, if an input file is specifically named in one EXCEPTION/ERROR procedure, there must not also be an EXCEPTION/ERROR procedure for all INPUT files.

IBM Extension

The file-name phrase is also valid for TRANSACTION files.

End of IBM Extension

INPUT Phrase

This phrase is valid for sequential, indexed, and relative files. When this phrase is specified, the procedure is applicable to all files opened in INPUT mode.

OUTPUT Phrase

This phrase is valid for sequential, indexed, and relative files. When this phrase is specified, the procedure is applicable to all files opened in OUTPUT mode.

I-O Phrase

This phrase is valid for sequential, indexed, and relative files. When this phrase is specified, the procedure is applicable to all files opened in I-O mode.

IBM Extension

The I-O phrase is also valid for TRANSACTION files.

End of IBM Extension

EXTEND Phrase

This phrase is valid for sequential files only. When this phrase is specified, the procedure is applicable to all files opened in EXTEND mode.

General Considerations

The EXCEPTION/ERROR Declaratives procedure is executed when one of the following conditions exists:

- After completing the standard system input/output error routine
- Upon recognition of an INVALID KEY or AT END condition when an INVALID KEY or AT END option has not been specified in the input/output statement
- When Status Key 1 is not equal to 0 following an I-O operation.

The EXCEPTION/ERROR Declarative procedures are performed when an input/output error occurs during execution of a READ, WRITE, REWRITE, START, DELETE, ACQUIRE, DROP, OPEN, or CLOSE statement. For

example, these procedures are activated when an input/output statement fails on a file that is in the open status.

After execution of the EXCEPTION/ERROR Declarative procedure, control is returned to the statement immediately following the input/output statement that caused the error.

Within a Declarative procedure, there must be no reference to any nondeclarative procedure. In the nondeclarative portion of the program, there must be no reference to procedure-names that appear in an EXCEPTION/ERROR Declarative procedure, except that PERFORM statements can refer to an EXCEPTION/ERROR Declarative procedure or to procedures associated with it.

Within an EXCEPTION/ERROR Declarative procedure, no statement can be executed that causes execution of a USE statement that has been previously invoked and has not yet returned control to the invoking routine.

Programming Notes

EXCEPTION/ERROR Declarative procedures can be used to check the status key values whenever an input/output error occurs. Additional information about the file causing the error can be obtained by using data from the mnemonic-names OPEN-FEEDBACK and I-O-FEEDBACK.

Care should be used in specifying EXCEPTION/ERROR Declarative procedures for any file. Prior to successful completion of an initial OPEN for any file, the current Declarative has not yet been established by the object program. Therefore, if any other I-O statement is executed for a file that has never been opened, no Declarative can receive control. However, if this file has been previously opened, the last previously established Declarative procedure receives control.

For example, an OPEN OUTPUT statement establishes a Declarative procedure for this file, and the file is then closed without error. During later processing, if a logic error occurs, control will go to the Declarative procedure established when the file was opened OUTPUT.

If there is no applicable USE procedure in the program when an I-O error occurs, execution can continue. Unless the program is terminated, or some other action taken, other errors may occur, causing undesirable results.

Conditional Statements

A conditional statement specifies that a truth value of a condition is to be determined, and that the subsequent action of the object program depends on this truth value. Figure 5-2 gives a list of the conditional statements.

Only the IF statement is discussed in this section; the other conditional statements are discussed elsewhere in this manual.

IF Statement

The IF statement causes a condition to be evaluated, and provides for alternative actions in the program, depending on that value.

Format

$$\text{IF condition } \boxed{\text{THEN}} \left\{ \begin{array}{l} \text{statement-1} \\ \text{NEXT SENTENCE} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{ELSE statement-2} \\ \text{ELSE NEXT SENTENCE} \end{array} \right\} \right]$$

IBM Extension

THEN is used as a separator.

End of IBM Extension

Statement-1 or statement-2 can be any one of the following:

- An imperative statement
- A conditional statement
- An imperative statement followed by a conditional statement.

The scope of an IF statement can be terminated by any of the following:

- A separator period
- If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting.

If the condition tested is true, one of the following actions takes place:

- Statement-1, if specified, is executed. If statement-1 contains a procedure branching statement, control is transferred according to the rules for that statement. If statement-1 does not contain a procedure-branching statement, the ELSE phrase, if specified, is ignored, and control passes to the next executable sentence.
- NEXT SENTENCE, if specified, is executed; that is, the ELSE phrase, if specified, is ignored, and control passes to the next executable sentence.

If the condition tested is false, one of the following actions take place:

- ELSE statement-2, if specified, is executed. If statement-2 contains a procedure-branching statement, control is transferred according to the rules for that statement. If statement-2 does not contain a procedure-branching statement, control is passed to the next executable sentence.

- ELSE NEXT SENTENCE, if specified, is executed. Therefore, statement-1, if specified, is ignored; control passes to the next executable sentence.
- If ELSE phrase is omitted, control passes to the next executable sentence.
- The ELSE NEXT SENTENCE phrase can be omitted if it immediately precedes the period that ends the conditional sentence.

Note: When the ELSE phrase is omitted, all statements following the condition and preceding the period for the sentence are considered to be part of statement-1.

Nested IF Statements

The presence of one or more IF statements within an initial IF statement constitutes a nested IF statement.

Statement-1 and statement-2 in IF statements can consist of one or more imperative statements and/or a conditional statement. If an IF statement appears as statement-1 or as part of statement-1, it is said to be nested. Nesting statements is much like specifying subordinate arithmetic expressions enclosed in parentheses and combined in large arithmetic expressions.

IF statements contained within IF statements must be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered must be considered to apply to the immediately preceding IF that has not already paired with an ELSE.

Figure 5-14 shows the possible true/false combinations for the following nested IF statement:

```

IF condition-1
  statement-1-1
  IF condition-2
    IF condition-3
      statement-3-1
    ELSE
      statement-3-2
  ELSE
    statement-2-2
  IF condition-4
    IF condition-5
      statement-5-1
    ELSE
      statement-5-2.

```

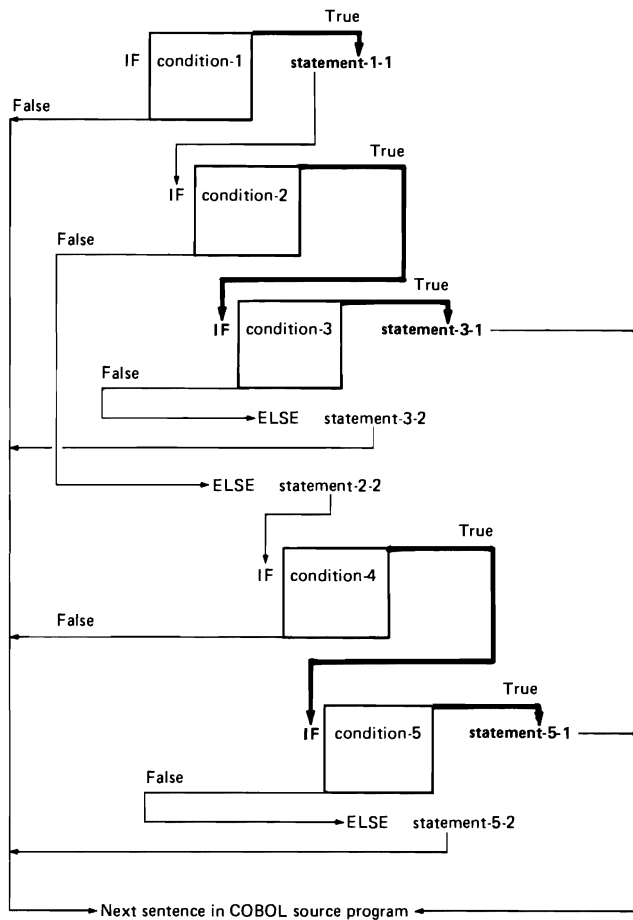


Figure 5-14. Nested IF Statement – True/False Combinations

Programming Notes: Because their logic is often difficult to follow, nested IF statements should, wherever possible, be avoided in a COBOL program. Often a series of simple IF statements can be used in place of the nested IF statement.

For example, the following series of simple IF statements give results equivalent to those achieved using the preceding nested IF statement example:

```
IF condition-1 NEXT SENTENCE
  ELSE GO TO PARA-2.
```

```
statement-1-1.
```

```
IF condition-2 NEXT SENTENCE
  ELSE GO TO PARA-1.
```

```
IF condition-3 statement-3-1 GO TO PARA-2
  ELSE statement-3-2 GO TO PARA-2.
```

PARA-1.

statement-2-2.

IF condition-4 NEXT SENTENCE
ELSE GO TO PARA-2.

IF condition-5 statement-5-1
ELSE statement-5-2.

PARA-2.

next-executable-statement.

Notice that Figure 5-14 also illustrates the logic flow for the preceding series of simple IF statements.

Input/Output Statements

COBOL input/output statements transfer data to and from files. In COBOL, the unit of data made available to the program is a record, and the COBOL user need be concerned only with such records. Provision is automatically made for such operations as the movement of data into buffers and/or internal storage, validity checking, error correction (when feasible), and unblocking and blocking of records.

The description of the file in the Environment Division and the Data Division governs which input/output statements are allowed in the Procedure Division.

All TRANSACTION file formats of the input/output statements are discussed in Chapter 7.

For information about COBOL file processing in relation to System/38 file processing, see Chapter 9. See Appendix I for a file structure support summary.

Common Input/Output Phrases

There are several phrases and concepts common to input/output statements. These are: status key, INVALID KEY condition, INTO/FROM identifier phrase, and current record pointer. The description of these phrases precedes the descriptions of the individual statements.

Status Key

If the FILE STATUS clause is specified in the file-control entry, a value is placed in the specified status key (the 2-character data item named in the FILE STATUS clause) during execution of any request on that file; the value indicates the status of that request. The value is placed in the status key before execution of any EXCEPTION/ERROR Declarative or INVALID KEY/AT END phrase associated with the request.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Appendix I.

INVALID KEY Condition

The INVALID KEY condition can occur during execution of a START, READ, WRITE, REWRITE, or DELETE statement. When the INVALID KEY condition is recognized, the actions are taken in the following order:

1. If the FILE STATUS clause is specified in the file-control entry, a value is placed into the status key to indicate an INVALID KEY condition (see Appendix I).
2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any EXCEPTION/ERROR Declarative procedure specified for this file is not performed.
3. If the INVALID KEY phrase is not specified, but an EXCEPTION/ERROR Declarative procedure is specified for the file, the EXCEPTION/ERROR procedure is executed.

When an INVALID KEY condition occurs, the input/output statement that caused the condition is unsuccessful. If the INVALID KEY phrase is not specified for a file, an EXCEPTION/ERROR procedure must be specified.

INTO/FROM Identifier Phrase

This phrase is valid for READ, REWRITE, and WRITE statements. The identifier specified must be the name of an entry in the Working-Storage Section, the Linkage Section, or of a record description for another previously opened file. Record-name/file-name and identifier must not refer to the same storage area. In both phrases, an implicit move is executed according to MOVE statement rules without the CORRESPONDING phrase.

The following illustrate the use of the INTO/FROM identifier phrase in an input/output statement:

READ file-name RECORD INTO identifier.

WRITE record-name FROM identifier.

Current Record Pointer

The current record pointer identifies a particular record accessed by a sequential input request. The record identified depends on the statement being executed. The OPEN, READ, RETURN, ROLLBACK, and START statements position the current record pointer as follows:

- The OPEN statement positions the current record pointer to the first record in the file.

IBM Extension

The current record pointer can be positioned to any record in the file by using the POSITION parameter of the Override with Data Base File (OVRDBF) command.

End of IBM Extension

- For a sequential access READ statement, or a dynamic access READ NEXT statement, the following considerations apply:
 - If an OPEN or START statement positioned the current record pointer, the record identified by the current record pointer is made available. If this record does not exist, the next existing record is made available.
 - If a previous READ statement positioned the current record pointer, the current record pointer is updated to point to the next existing record in the file; that record is then made available.

IBM Extension

- For a dynamic access READ FIRST statement, the current record pointer is positioned to point to the first record in the file; that record is then made available.
- For a dynamic access READ LAST statement, the current record pointer is positioned to point to the last record in the file; that record is then made available.
- For a dynamic access READ PRIOR statement, the current record pointer is positioned to point to the previous existing record in the file; that record is then made available.

End of IBM Extension

- For the RETURN statement, the following considerations apply:
 - The first RETURN statement positions the current record pointer to the first record in the file, and that record is then made available.
 - If a previous RETURN statement positioned the current record pointer, the current record pointer is updated to point to the next existing record in the file, and the record is then made available.

- For the ROLLBACK statement, the following considerations apply to any file under commitment control:
 - The ROLLBACK statement sets the current record pointer to the pointer's position at the previous commitment boundary. This is important to remember if you are doing sequential processing.
 - The current record pointer is set to the pointer's position at the OPEN if no COMMIT statement has been issued since the file was opened.
 - The current record pointer is undefined for any file under commitment control that is not open.
- The START statement positions the current record pointer to the first record in the file that satisfies the implicit or explicit comparison specified in the START statement.

The setting of the current record pointer is affected only by the OPEN, START, RETURN, READ, and ROLLBACK statements. The concept of the current record pointer has no meaning for files with an access mode of random, for TRANSACTION files, or for output files.

IBM Extension

DB-FORMAT-NAME Special Register

After the execution of an input/output statement, for a FORMATFILE or DATABASE file, the DB-FORMAT-NAME special register is modified according to the following rules:

- After completion of a successful READ, WRITE, REWRITE, START, or DELETE operation, the record format name used in the I-O operation is implicitly moved to the special register.
- After an unsuccessful input/output operation, DB-FORMAT-NAME contains the record format name used in the last successful input/output operation.
- DB-FORMAT-NAME is implicitly defined as PICTURE X(10).

End of IBM Extension

ACCEPT Statement

The function of the ACCEPT statement is to obtain low volume data. ACCEPT statement execution causes the transfer of data into the specified identifier. There is no editing or error checking of the incoming data. The formats of the ACCEPT statement are as follows:

Format 1

ACCEPT identifier [FROM mnemonic-name]

Format 2

ACCEPT identifier FROM {
DATE
DAY
TIME }

Format 3 – Feedback

ACCEPT identifier FROM mnemonic-name
[FOR file-name]

Format 4 – Local Data Area

ACCEPT identifier-1 FROM mnemonic-name
[FOR { identifier-2
literal }]

IBM Extension

Format 5 – TRANSACTION Attributes

See Chapter 7 for information about this format.

End of IBM Extension

Format 1 Considerations

This format is used to transfer data from an input/output device to the identifier. Identifier can be any fixed length group item, or an elementary alphabetic, alphanumeric, or zoned decimal item.

When the FROM phrase is omitted, the ACCEPT statement obtains input from the job input stream for batch jobs, and from the work station for interactive jobs.

The job input stream is CL request data. If there is no data in the input stream, an exception occurs. See “How to Execute a COBOL Program” in Chapter 8 for further information on the placement of input data for a batch job.

When the FROM phrase is specified, mnemonic-name must be associated with an input/output device that is specified in the SPECIAL-NAMES paragraph. The input/output device can be the work station (REQUESTOR) or the system operator’s station (CONSOLE or SYSTEM-CONSOLE). If mnemonic-name is REQUESTOR and the job is a batch job, the job input stream is used.

When the input is from the job input stream, the following rules apply:

- An input record size of 80 characters is assumed.
- If the identifier is up to 80 characters in length, the input data must appear as the first character within the input record. Any characters beyond the length of identifier are truncated.
- If the identifier is longer than 80 characters, succeeding input records are read until the storage area of the identifier is filled. If the length of the identifier is not an exact multiple of 80 characters, the last input record is truncated.

When the device is the work station, the input record size is 100. When the device is console, the input record size is 58. The following steps occur:

1. A system-generated inquiry message containing the program-name, the text “AWAITING REPLY FOR POSITION(S)”, and the beginning and ending positions is automatically sent to the system or work station operator. Previous DISPLAYs can also appear on the ACCEPT screen.
2. Execution is suspended.
3. The reply is moved into the identifier, and execution is resumed after a reply is made by the system operator to the inquiry in step 1. The reply value is made available to the program as it was entered, in uppercase or lowercase.
4. If the identifier is longer than the input record size, then succeeding input records are read (steps 1-3) until the identifier is filled.

If the incoming reply is longer than the identifier, the character positions beyond the length of identifier are truncated.

The source of input data is dependent upon the type of program initiation as follows:

Method of Program Initiation	Mnemonic-Name Associated with SYSTEM-CONSOLE	Mnemonic-Name Associated with REQUESTOR	Data Source When FROM Phrase Omitted
BATCH	System operator's station	Job input stream	Job input stream
INTERACTIVE	System operator's station	Work station	Work station

Format 2 Considerations

This format is used to transfer the system date or system time to the identifier, using the rules for the MOVE statement without the CORRESPONDING phrase. Identifier can be a group item, or an elementary alphanumeric, alphanumeric edited, numeric, or numeric edited item.

IBM Extension

A numeric item can also be defined as COMPUTATIONAL-3 or COMPUTATIONAL-4.

End of IBM Extension

DATE, DAY, and TIME implicitly have USAGE DISPLAY.

DATE has the implicit PICTURE 9(6). The sequence of data elements from left to right is: two digits for year of century, two digits for month of year, two digits for day of month. Thus July 4, 1976 is expressed as 760704.

DAY has the implicit PICTURE 9(5). The sequence of data elements from left to right is: two digits for year of century, three digits for day of year. Thus, July 4, 1976 is expressed as 76186.

TIME has the implicit PICTURE 9(8). The sequence of data elements from left to right is: two digits for hour of day, two digits for minute of hour, two digits for second of minute, two digits for hundredths of second. Thus 12.25 seconds after 2:41 p.m. is expressed as 14411225.

Format 3 Considerations

This format is used to transfer feedback information from an active file to the identifier. The identifier can be any fixed-length group item or an elementary alphabetic, alphanumeric, or zoned decimal item. The file must be defined in an FD entry, and must be open prior to the execution of the ACCEPT statement. If the file is not open, the contents of identifier remain unchanged.

The FROM phrase specifies a mnemonic-name that must be associated with a function-name of OPEN-FEEDBACK or I-O-FEEDBACK in the SPECIAL-NAMES paragraph.

When the FOR phrase is specified, the feedback information is from the file specified in the phrase. When the FOR phrase is not specified, the feedback information is from the last file opened or used in an input or output operation.

See Appendix I for a discussion of the I-O-FEEDBACK and OPEN-FEEDBACK areas. See the *CPF Programmer's Guide* for a layout and description of the data areas contained in the feedback areas.

Format 4 Considerations

This format is used to transfer data to identifier-1 from the system-defined local data area created for a job.

This format is only applicable when the mnemonic-name in the SPECIAL-NAMES paragraph is associated with the function-name LOCAL-DATA.

The move into identifier-1 takes place according to the rules for the MOVE statement for a group move without the CORRESPONDING phrase.

When the FOR phrase is specified, it is syntax checked during compilation but treated as comments during execution. The value of literal or identifier-2 indicates the program device name of the device that is associated with the local data area. There is only one local data area for each job, and all devices in a job access the same local data area. Literal, if specified, must be nonnumeric and 10 characters or less in length. Identifier-2, if specified, must refer to an alphanumeric data item, 10 characters or less in length.

See "Local Data Area" in Chapter 9 for more information.

Programming Notes

The Format 1 ACCEPT statement is useful for exceptional situations in a program when operator intervention (to supply a given message, code, or exception indicator) is required. The operator must, of course, be supplied with the appropriate messages with which to reply.

The Format 2 ACCEPT statement allows the programmer access to the current date (in two formats) and time of day, as carried by the system. This can be useful in identifying when a particular run of a program was executed. It can also be used to supply the date in headings, footings, and so on.

ACQUIRE Statement

IBM Extension

The ACQUIRE statement acquires a program device for a TRANSACTION file.

See Chapter 7 for the format, and for a discussion of this statement.

End of IBM Extension

CLOSE Statement

The CLOSE statement terminates the processing of volumes and files. REWIND, LOCK, and REMOVAL phrases are specified, as applicable. The formats of the CLOSE statement are as follows:

Format 1

$$\begin{array}{l} \text{CLOSE file-name-1} \left[\begin{array}{l} \left\{ \begin{array}{l} \text{REEL} \\ \text{UNIT} \end{array} \right\} \left[\begin{array}{l} \text{WITH NO REWIND} \\ \text{FOR REMOVAL} \end{array} \right] \\ \text{WITH} \left\{ \begin{array}{l} \text{NO REWIND} \\ \text{LOCK} \end{array} \right\} \end{array} \right] \\ \left[\begin{array}{l} \text{, file-name-2} \left[\begin{array}{l} \left\{ \begin{array}{l} \text{REEL} \\ \text{UNIT} \end{array} \right\} \left[\begin{array}{l} \text{WITH NO REWIND} \\ \text{FOR REMOVAL} \end{array} \right] \\ \text{WITH} \left\{ \begin{array}{l} \text{NO REWIND} \\ \text{LOCK} \end{array} \right\} \end{array} \right] \dots \end{array} \right]$$

Format 2

See Chapter 7 for a discussion of the TRANSACTION file format.

Each file-name designates a file upon which the CLOSE statement is to operate. These files:

- Need not have the same organization or access mode
- Must not be sort or merge files.

A CLOSE statement without the REEL/UNIT phrase can be successfully executed for a file. In this case, an OPEN statement for the file must be executed before any other input/output statement can refer explicitly or implicitly to the file. This is true for all input/output statements except a SORT/MERGE statement with the USING or GIVING phrases.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the CLOSE statement is executed.

If the file is open and the execution of a CLOSE statement is unsuccessful, the EXCEPTION/ERROR procedure for the file, if specified, is executed.

If a CLOSE statement for an open file is not executed before a STOP RUN for this run unit, the file is implicitly closed.

If a CANCEL statement is executed for a program with an open file:

- The status of that file is unpredictable
- The file can be logically damaged

- The file can keep the allocated device longer than necessary.

If the SELECT OPTIONAL clause is specified in the file-control entry for this file and the file is not present at execution time, standard end-of-file processing is not performed.

The following tables illustrate organization, access, device, and volume considerations for the CLOSE statement. The letter codes used in the tables are defined in the section following the tables.

Sequential Organization

Access	SEQUENTIAL									
	Device	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	DISKETTE	DISK	DATABASE	FORMATFILE
CLOSE	K, J	K, J	K, J	K, J	K, J	K, J	K, J	K, J	K, J	K, J
CLOSE WITH LOCK	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E
REEL/UNIT	—	—	—	—	—	—	—	—	—	—
REMOVAL	—	—	—	—	—	—	—	—	—	—
NO REWIND	—	—	—	—	—	—	—	—	—	—

Sequential Organization

Access	SEQUENTIAL	
	TAPEFILE	
	Volume	Single
CLOSE	K, J, G	K, J, G, A
CLOSE WITH LOCK	K, J, G, E	K, J, G, A, E
CLOSE NO REWIND	K, J, B	K, J, B, A
CLOSE REEL/UNIT	C	K, F, G
CLOSE REEL/UNIT FOR REMOVAL	C	K, F, D, G
CLOSE REEL/UNIT WITH NO REWIND	C	K, F, B

Indexed Organization

Access	Any	
	Device	DISK
CLOSE	K, J	K, J
CLOSE WITH LOCK	K, J, E	K, J, E
REEL/UNIT	—	—
REMOVAL	—	—
NO REWIND	—	—

Relative Organization

	Access	Any	
	Device	DISK	DATABASE
CLOSE		K, J	K, J
CLOSE WITH LOCK		K, J, E	K, J, E
REEL/UNIT		—	—
REMOVAL		—	—
NO REWIND		—	—

Letter Code Meaning

- An invalid combination.
- A No effect on any previous volumes. Any additional volumes are not processed.
- B The current volume is left in its present position. The reel is not rewound.

IBM Extension

The system always rewinds and unloads the tape when REEL/UNIT is specified on the CLOSE statement.

End of IBM Extension

- C Optional, but only syntax-checked (performs no function at execution time).
- D The current volume is rewound and unloaded. The system is notified that the volume is logically removed from this run unit. However, the volume can be accessed again, after execution of a CLOSE statement without the REEL/UNIT phrase and an OPEN statement for this file.
- E COBOL ensures that this file cannot be reopened during this execution of the program.
- F Close volume procedures. The labels are handled as follows:

Label Records		
Mode of File	Standard	Omitted
INPUT	F01	F02
OUTPUT	F03	F04
I-O	F01	F02

F01 The current volume is positioned to read the labels. If this is the last volume for the file, the next executed READ statement receives the AT END condition. If this is not the last volume, the following actions are taken:

1. The current volume is unloaded
2. The beginning standard labels on the next volume are read
3. The next executed READ statement gets the first record on the newly mounted volume.

F02 The current volume is unloaded. If all of the reels as specified on the REELS parameter of the Create Tape File (CRTTAPF) or Override with Tape File (OVRTAPF) CL command have been processed, the next executed READ statement receives the AT END condition. If there are more reels, the next volume is mounted, and the next executed READ statement gets the first record on the newly mounted volume.

F03 The standard end-of-volume labels are written. The next volume is mounted. The standard beginning labels are written on the new volume. The next executed WRITE statement places the next logical record on the newly mounted volume.

F04 The system end-of-volume procedures for nonlabeled tapes are executed. The next volume is mounted. The system beginning of volume procedures for nonlabeled tapes are executed. The next executed WRITE statement places the next logical record on the newly mounted volume.

G The current volume is positioned at its beginning.

J The record area associated with the file-name is no longer available after successful execution of this statement. Unsuccessful execution of this statement leaves availability of the record data area undefined.

Labels are processed as follows:

	Label Records	
Mode of File	Standard	Omitted
INPUT	J01	J02
OUTPUT	J03	J04
I-O	J01	J02

J01 If the file is positioned at its end, the label records are read and verified, and the file is closed. If the file is not at its end, the file is closed.

J02 The file is closed.

J03 The standard label records are written, and the file is closed.

J04 The file is closed without any label processing.

K May be executed only for an open file.

COMMIT Statement

The COMMIT statement provides a way of synchronizing changes to data base records while preventing other jobs from modifying those records until the COMMIT is performed. The format of the COMMIT statement is:

Format

```

COMMIT

```

When the COMMIT statement is executed, all changes made to files under commitment control since the previous commitment boundary are made permanent. A commitment boundary is established by the successful execution of a ROLLBACK or COMMIT statement. If no COMMIT or ROLLBACK has been issued in the current job, a commitment boundary is established by the first OPEN of any file under commitment control in the job. Changes are made to all files under commitment control in the job, not just to files under commitment control in the COBOL program that issues the COMMIT statement.

When a COMMIT is executed, all record locks held by the job since the last commitment boundary for files under commitment control are released and the records become available to other jobs.

The COMMIT statement only affects files under commitment control. If a COMMIT is executed and there are no files opened under commitment

control, the COMMIT statement has no effect and no commitment boundary is established.

The COMMIT statement does *not*:

- Modify the I-O-FEEDBACK area for any file
- Change the current record pointer for any file
- Set a file status value for any file.

For more information on commitment control, see “Commitment Control Considerations” in Chapter 9.

DELETE Statement

The DELETE statement logically removes a record from an indexed or relative file. The format of the DELETE statement is as follows:

Format

DELETE file-name RECORD **[FORMAT IS { identifier }
literal]**
[INVALID KEY imperative-statement]

After successful execution of a DELETE statement, the record is logically removed from the file. It is no longer accessible. Execution of the DELETE statement does not affect the contents of the record area associated with file-name.

If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the DELETE statement is executed.

The current record pointer is not affected by the execution of the DELETE statement.

The following tables illustrate organization, access, and device considerations for the DELETE statement. The letter codes used in the tables are defined in the section following the tables.

Sequential Organization

	Device	Any
	Access	SEQUENTIAL
DELETE Verb		Not allowed

Relative Organization

Device	DISK			DATABASE		
	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
DELETE Verb	A, P, Z	B, P, Z	B, P, Z	A, P, Z	B, P, Z	B, P, Z
INVALID KEY	—	O, U	O, U	—	O, U	O, U
FORMAT	—	—	—	—	—	—

Indexed Organization

Device	DISK			DATABASE		
	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
DELETE Verb	A, P, Z	D, P, Z	D, P, Z	A, P, Z	D, P, Z	D, P, Z
INVALID KEY	—	O, U	O, U	—	O, U	O, U
FORMAT	—	—	—	—	S, F	S, F

Letter Code Meaning

- An invalid combination.
- A The last input/output statement must have been a successfully executed READ statement. When the DELETE statement is executed, the system logically removes the record retrieved by that READ statement.
- B The system logically removes the record identified by the contents of the RELATIVE KEY data item. If the file does not contain such a record, an INVALID KEY condition exists. The space is then available for a new record with the same RELATIVE KEY value.
- D The system logically removes the record identified by the contents of the RECORD KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

IBM Extension

When EXTERNALLY-DESCRIBED-KEY is specified for the file, the key field in the record area⁷ for the format specified by the FORMAT

⁷ The key field in the record area is the location in the buffer selected in accordance with a record format or specification in order to build a search argument.

phrase is used to find the record to be deleted. If the FORMAT phrase is not specified, the first format defined in the program for the file is used to find the record to be deleted.

If the DUPLICATES phrase was specified for this file, the last input/output statement executed for this file before execution of the DELETE statement must have been a successfully executed READ statement. The record read by that statement is the record that is deleted.

In this case, the FORMAT phrase is not used to find the record to be deleted. The READ statement is required to ensure that the proper record is deleted when there are duplicates. If a successful read operation did not occur before the delete operation:

- The file status key, if defined, is set to 94.
- The USE AFTER STANDARD EXCEPTION/ERROR procedure, if specified, is executed.
- The delete operation is not executed.

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format must be operated on.

The literal or identifier must be a character-string of 10 characters or less. If an identifier is specified, it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

_____ End of IBM Extension _____

- O Optional.
- P Allowed when the file is opened for I-O.
- S Required when processing a file that has multiple record formats and has unique keys.
- U The INVALID KEY phrase must be specified for files in which an applicable USE procedure is not specified.

IBM Extension

Z The action of this statement can be inhibited at program execution time by the inhibit write (INHVRT) parameter of the Override with Data Base File (OVRDBF) CL command. When this parameter is specified, non-zero file status codes are not set for data dependent errors. Duplicate key and data conversion errors are examples of data dependent errors.

See the *CL Reference Manual* for more information on this command.

End of IBM Extension

DISPLAY Statement

The DISPLAY statement transfers low-volume data to an output device.

Format 1

DISPLAY { identifier-1 } [, identifier-2]
 literal-1 [, literal-2]
 . . . [UPON mnemonic-name]

IBM Extension

Format 2—Local Data Area

DISPLAY { identifier-1 } [[, identifier-2]]
 literal-1 [, literal-2]]

UPON mnemonic-name

[FOR { identifier-3 }]
 literal-3]

End of IBM Extension

Format 1 Considerations

The DISPLAY statement transfers the contents of each operand to the output device in the left-to-right order in which the operands are listed. When a DISPLAY statement is executed, the data contained in the sending field is transferred to the output device. The size of the sending field is the total character count of all operands listed. If the total character count is less than the maximum logical record size, the remaining rightmost characters are padded with spaces. If the total character count exceeds the maximum, as many records are written as are needed to display all operands. Any operand being displayed when the end of a record is reached is continued in the next record.

Numeric identifiers not described as USAGE IS DISPLAY are converted automatically to zoned decimal.

IBM Extension

COMPUTATIONAL-4 items are also converted to zoned decimal. Signed noninteger numeric literals are allowed.

End of IBM Extension

Signed values in numeric fields cause the last character to show both the sign and number. For example, if SIGN WITH SEPARATE CHARACTER is not specified and two numeric items have the values -34 and 34, they are displayed as 3M and 34, respectively. If SIGN WITH SEPARATE CHARACTER is specified, a + or a - sign is displayed as either leading or trailing, depending on how the number was specified.

Programming Note: Group items containing packed or binary data (COMP, COMP-3, or COMP-4) should not be displayed on a display station. Such data can contain display station control characters which can cause undesirable and unpredictable results.

A literal can be any figurative constant except the ALL literal. When a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

When the UPON phrase is omitted, the DISPLAY statement sends output to the REQUESTOR. When the UPON phrase is specified, mnemonic-name must be associated in the SPECIAL-NAMES paragraph with either the work station (REQUESTOR) or the system operator's station (CONSOLE or SYSTEM-CONSOLE).

The record length depends on the device as follows:

Output	Maximum Logical Record Size
Job log	120 characters
Work station	58 characters
System console	58 characters

When a program in a batch job executes a DISPLAY statement without the UPON phrase, or with an UPON phrase associated with the REQUESTOR, the output is sent to the job log in an informational message of severity 80. You can change the severity of this message using the Change Message Description (CHGMSGD) CL command.

For more information, see the *CL Reference Manual*.

For an interactive job that uses display device files, DISPLAY statements are not normally used. If you do use them, the following considerations apply.

When an interactive job executes a DISPLAY statement, the logical record appears on the screen in the Program Messages display.

The following screen shows a sample Program Messages display.

System messages
for this session

Program messages
for this session

```

7/16/82 14:51:12 PROGRAM MESSAGES
JOB WS1.QPGMR.000745 STARTED 07/16/82 14:50:22.
SAMPLE PROGRAM MESSAGE FROM PREVIOUS EXECUTION.
SAMPLE PROGRAM MESSAGE FROM CURRENT EXECUTION.

```

This display contains messages from the current program execution, as well as messages relating to other activities in the session.

The display device file on the screen when a DISPLAY statement is executed determines whether program execution is suspended as a result of the DISPLAY statement execution.

- If the parameter RSTDSP(*NO) is specified when the display device file is changed or created (CHGDSPF or CRTDSPF command), DISPLAY statement execution suspends program execution, and the Program Messages display appears on the screen. You must press the Enter key to resume program execution. The previous display returns to the screen immediately.
- If the parameter RSTDSP(*YES) is specified when the display device file is changed or created (CHGDSPF or CRTDSPF command), DISPLAY statement execution does not suspend program execution. The Program Messages display appears on the screen, and remains on the screen until one of the following happens:
 - The program executes a nonsubfile READ or WRITE statement for that file. The Program Messages Display then disappears, and the display device file is returned to the screen.
 - The program terminates.

Programming Note: If you want to suspend program execution, code an ACCEPT statement after the DISPLAY statement. Program execution is suspended until you press the Enter key.

To view output records after the program terminates, press the CF7 key from the Command Entry display.

For additional information on interactive processing, see Chapter 7. For additional information on the RSTDSP parameter, see the CHGDSPF and CRTDSPF commands in the *CL Reference Manual*.

When a program started by a work station operator sends a DISPLAY to the system operator's station (separate from the work station), program execution is not suspended.

The location of the output data is dependent upon the type of program initiation as follows:

Method of Initiation	Mnemonic-Name Associated with SYSTEM-CONSOLE	Mnemonic-Name Associated with REQUESTOR	UPON Phrase Omitted
BATCH	System operator's station	Job log	Job log
INTERACTIVE	System operator's station	Work station	Work station

Format 2 Considerations

This format is used to transfer data to the system-defined local data area created for a job.

This format is only applicable when the mnemonic-name in the SPECIAL-NAMES paragraph is associated with the function name LOCAL-DATA.

The DISPLAY statement's literal operands, or the contents of the DISPLAY statement's identifier operands, are written to the system-defined local data area of the job containing the program that issues the DISPLAY. The data is written to the local data area according to the rules of the MOVE statement for a group move, without the CORRESPONDING phrase.

The FOR phrase, when specified, is syntax checked during compilation but is treated as comments during execution. The value of literal-3 or identifier-3 indicates the program device name of the device that is writing data to the local data area. There is only one local data area for each job, and all devices in a job access the same local data area. Literal-3, if specified, must be nonnumeric and 10 characters or less in length, and identifier-3, if specified, must refer to an alphanumeric data item 10 characters or less in length.

For more information, see "Local Data Area" in Chapter 9.

End of IBM Extension

DROP Statement

The DROP statement releases a program device that was acquired by a TRANSACTION file.

See Chapter 7 for the format, and for a discussion of this statement.

End of IBM Extension

OPEN Statement

The OPEN statement initiates file processing. It also checks and/or writes labels. The formats of the OPEN statement are as follows:

Format 1 – Sequential Files

$$\text{OPEN} \left\{ \begin{array}{l} \text{INPUT file-name-1} \left[\begin{array}{l} \text{REVERSED} \\ \text{WITH NO REWIND} \end{array} \right] \\ \left[, \text{file-name-2} \left[\begin{array}{l} \text{REVERSED} \\ \text{WITH NO REWIND} \end{array} \right] \right] \dots \\ \text{OUTPUT file-name-3} \left[\text{WITH NO REWIND} \right] \\ \left[, \text{file-name-4} \left[\text{WITH NO REWIND} \right] \right] \dots \\ \text{I-O file-name-5} \left[, \text{file-name-6} \right] \dots \\ \text{EXTEND file-name-7} \left[, \text{file-name-8} \right] \dots \end{array} \right\} \dots$$

Format 2 – Indexed and Relative Files

$$\text{OPEN} \left\{ \left\{ \begin{array}{l} \text{INPUT} \\ \text{OUTPUT} \\ \text{I-O} \end{array} \right\} \text{file-name-1} \left[, \text{file-name-2} \right] \dots \right\} \dots$$

Format 3 – TRANSACTION Files

See Chapter 7 for a discussion of this format.

Each file-name designates a file upon which the OPEN statement is to operate. The files specified need not have the same organization or access. Each file-name must be defined in an FD entry in the Data Division, and must not name a sort or merge file. The FD entry must be compatible with the information supplied to the system when the file was defined.

At least one of the phrases (INPUT, OUTPUT, I-O, or EXTEND) must be specified. These phrases can appear in any order. More than one file name can be specified in each phrase.

A file can be opened for INPUT, OUTPUT, I-O, or EXTEND in the same program. After the first OPEN statement is executed for a file, each subsequent OPEN statement execution must be preceded by a successful CLOSE file statement execution without the LOCK phrase.

The following tables illustrate organization, access, and device considerations for the OPEN statement. The letter codes used in the tables are defined in the section following the tables.

Sequential Organization

Access	SEQUENTIAL									
	Device	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	TAPEFILE	DISKETTE	DISK	DATABASE
OPEN Verb	S	S	S	S	S	L, S	S	S	S, C	S
INPUT	R, A, F	—	—	—	—	—	O, A, F, L1, N	O, A, F, N	O, A, K, F, N	O, A, K, F, N
OUTPUT	—	R, J	R, J	R, J	R, J	—	O, J, L2, N	O, J, N	O, G, N	O, G
I-O	—	—	—	—	—	—	—	—	O, M, K	O, M, K
NO REWIND	—	—	—	—	—	—	O, D	—	—	—
REVERSED	—	—	—	—	—	—	O, B	—	—	—
EXTEND	—	—	—	—	—	—	O, E, L3	—	O, E	O, E

Relative Organization

Device	DISK			DATABASE			
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
OPEN Verb	H, S	H, S	H, S	H, S	H, S, C	H, S, C	H, S, C
INPUT	O, A, K, N	O, A	O, A, K	O, A, K	O, A, K, N	O, A	O, A, K
OUTPUT	O, G, N	O, G	O, G	O, G	O, G, N	O, G	O, G
I-O	O, M, K	O, M	O, M, K	O, M, K	O, M, K	O, M	O, M, K
NO REWIND	—	—	—	—	—	—	—
REVERSED	—	—	—	—	—	—	—
EXTEND	—	—	—	—	—	—	—

Indexed Organization

Device	DISK			DATABASE			
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
OPEN Verb	S	S	S	S	S, C	S, C	S, C
INPUT	O, A, K, N	O, A	O, A, K	O, A, K	O, A, K, N	O, A	O, A, K
OUTPUT	O, G, N	O, G	O, G	O, G	O, G, N	O, G	O, G
I-O	O, M, K	O, M	O, M, K	O, M, K	O, M, K	O, M	O, M, K
NO REWIND	—	—	—	—	—	—	—
NO REVERSED	—	—	—	—	—	—	—
EXTEND	—	—	—	—	—	—	—

**Letter
Code Meaning**

- An invalid combination.
- A The file is opened for input operations. The current record pointer is set to the first record in the file. If no records exist in the file, the current record pointer is set so that execution of the first sequential READ statement results in an AT END condition.
- B OPEN statement execution positions the file at its end. Subsequent READ statements make the data records available in reverse order, starting with the last record. REVERSED can only be specified for input files.

IBM Extension

- C The file may be placed under commitment control.

See "Commitment Control Considerations" in Chapter 9 for more information.

End of IBM Extension

- D The OPEN statement does not reposition the file. The tape must be positioned at the beginning of the desired file before execution of the OPEN statement.

IBM Extension

The system keeps track of the current position on the tape and automatically positions the tape to the proper place. When processing a multfile tape volume, all CLOSE statements should specify the LEAVE phrase. When the next file on the volume is opened, the system determines which direction the tape should be moved to most efficiently get to the desired file.

End of IBM Extension

- E The EXTEND phrase permits opening the file for output operations. OPEN EXTEND statement execution prepares the file for the addition of records. These additional records immediately follow the last record in the file. Subsequent WRITE statements add records as if the file had been opened for OUTPUT. The EXTEND phrase can be specified when a file is being created.
- F If SELECT OPTIONAL is specified in the file-control entry, OPEN statement execution causes the program to check for the presence or absence of this file at execution time. If the file is absent, the first READ statement for this file causes the AT END condition to occur.

IBM Extension

G Only a physical file is cleared when opened for OUTPUT. When the file is successfully opened, it contains no records. If an attempt is made to open a logical file for OUTPUT, the file is opened but no records are deleted. The file is treated as though the EXTEND phrase had been specified. To clear a logical file, all the members on which the logical file is based should be cleared.

End of IBM Extension

H Not allowed for logical file members:

- That are based on more than one physical file.
- That contain select/omit logic.

I Allowed when the file is opened for INPUT.

J The file is opened to allow only output operations. When the file is successfully opened, it contains no records.

IBM Extension

K The first record to be made available to the program can be specified at execution time by using the POSITION parameter on the OVRDBF CL command. See the *CL Reference Manual* for more information on this command.

End of IBM Extension

L When label records are specified but not present, or when label records are present but not specified, execution of the OPEN statement can have unpredictable results.

L1 The beginning labels are checked.

L2 The labels are checked, then new labels are written.

L3 The following results occur:

- Beginning file labels are processed only if this is a single-volume file.
- Beginning volume labels of the last existing volume are checked.
- The file is positioned to the existing ending file labels. The labels are checked and then deleted.
- Processing continues as if the file were opened as an output file.

- M The file is opened for both input and output operations. The current record pointer is set to the first record in the file. If no records exist in the file, the current record pointer is set so that execution of the first sequential READ statement results in an AT END condition.
- N The compiler generates code to block output records or unblock input records if the conditions listed in “Unblocking Input Records and Blocking Output Records” in Chapter 9 are satisfied.
- O Optional.
- R Required.
- S The successful execution of an OPEN statement determines the availability of the file and results in that file being open. Before successful execution of the OPEN statement for a file, no statement, except a SORT or MERGE statement with the USING or GIVING phrase, that refers explicitly or implicitly to that file can be executed. The successful execution of the OPEN statement makes the associated record area available to the program. It does not obtain or release a data record.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the OPEN statement is executed.

If an OPEN statement is issued for a file that is already open, the EXCEPTION/ERROR procedure for this file, if specified, is executed. See Appendix I for the file status codes.

READ Statement

At execution time, the READ statement makes a record available before execution of any statement following the READ statement.

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a file.

The formats for the READ statement are as follows:

Format 1 – Sequential Retrieval Using SEQUENTIAL Access

READ file-name RECORD

[INTO identifier-1]

[FORMAT IS { identifier-2 }
literal-1]

[AT END imperative-statement]

Format 2 – Sequential Retrieval Using DYNAMIC Access

READ file-name { FIRST
LAST } RECORD
NEXT
PRIOR

[INTO identifier-1]

[FORMAT IS { identifier-2 }
literal-1]

AT END imperative-statement

Format 3 – Random Retrieval

READ file-name RECORD [INTO identifier-1]

[FORMAT IS { identifier-2 }
literal-1]

[INVALID KEY imperative-statement]

Format 4 – TRANSACTION File (Nonsubfile)

Format 5 – TRANSACTION File (Subfile)

See Chapter 7 for a discussion of these formats.

File-name must be defined in a Data Division FD entry, and must not name a sort or merge file. If more than one record-description entry is associated

with file-name, these records automatically share the same storage area. That is, they are implicitly redefined.

After a READ statement is executed, only those data items within the range of the current record are replaced. Data items stored beyond this range are undefined. Figure 5-15 illustrates this concept.

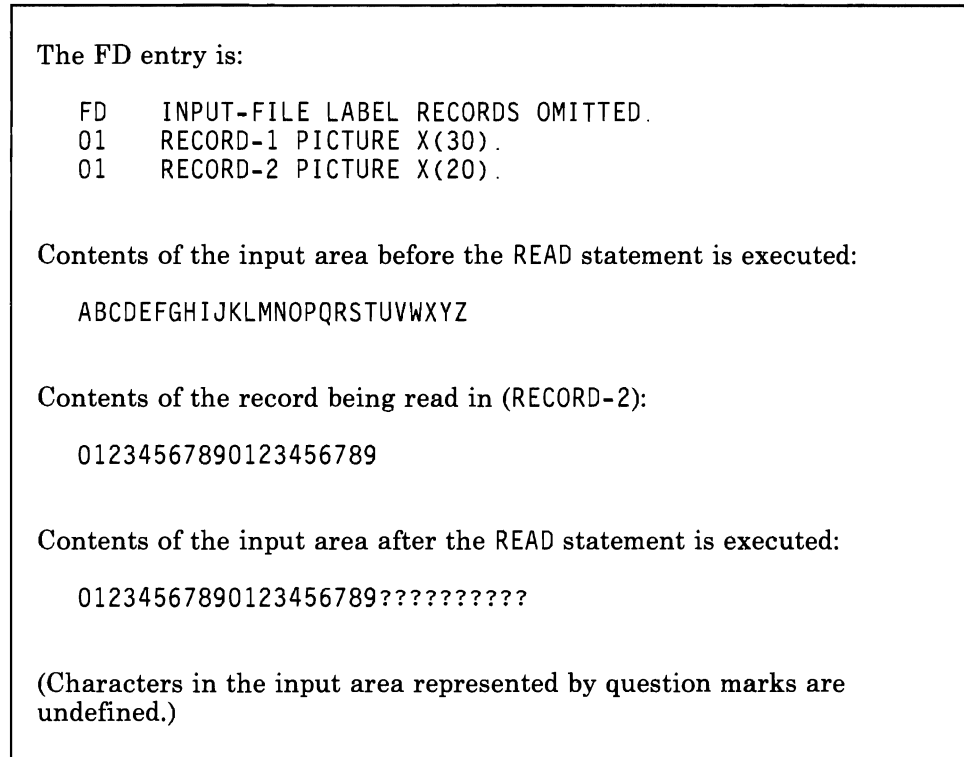


Figure 5-15. READ Statement with Multiple Record Descriptions

The following tables illustrate organization, access, and device considerations for the READ statement. The letter codes used in the tables are defined in the section following the tables.

Sequential Organization

Access	SEQUENTIAL										
	Device	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	TAPEFILE	DISKETTE	DISK	DATABASE	FORMATFILE
READ Verb	A, I, G1, N	—	—	—	—	—	A, I, G1, N, V	A, I, G1, N, V	A, I, P, G1, N	A, I, P, G1, N	A, I, G1, N
NEXT	—	—	—	—	—	—	—	—	—	—	—
LAST	—	—	—	—	—	—	—	—	—	—	—
FIRST	—	—	—	—	—	—	—	—	—	—	—
PRIOR	—	—	—	—	—	—	—	—	—	—	—
INTO	O, B	—	—	—	—	—	O, B	O, B	O, B	O, B	O, B
AT END	O, E, D1	—	—	—	—	—	O, E, D1	O, E, D1	O, E, D1	O, E, D1	O, E, D1
INVALID KEY	—	—	—	—	—	—	—	—	—	—	—
FORMAT	—	—	—	—	—	—	—	—	—	—	—

Relative Organization

Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM
READ Verb	A, I, P, G2	A, I, P, G3	A, I, P	A, I, P, G2	A, I, P, G3	A, I, P
NEXT	—	—	O, Z1	—	—	O, Z1
FIRST	—	—	—	—	—	—
LAST	—	—	—	—	—	—
PRIOR	—	—	—	—	—	—
INTO	O, B	O, B	O, B	O, B	O, B	O, B
AT END	O, E, D2	—	O, E, D2	O, E, D2	—	O, E, D2
INVALID KEY	—	O, U	O, U	—	O, U	O, U
FORMAT	—	—	—	—	—	—

Indexed Organization

Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM
READ Verb	A, I, P, G4	A, I, P, G5	A, I, P	A, I, P, G4	A, I, P, G5	A, I, P
NEXT	—	—	O, Z2	—	—	O, Z3
FIRST	—	—	—	—	—	O, Z3
LAST	—	—	—	—	—	O, Z3
PRIOR	—	—	—	—	—	O, Z3
INTO	O, B	O, B	O, B	O, B	O, B	O, B
AT END	O, E, D2	—	O, E, D2	O, E, D2	—	O, E, D2
INVALID KEY	—	O, U	O, U	—	O, U	O, U
FORMAT	—	—	—	O, F, X	O, F, W	O, F, Y

**Letter
Code Meaning**

- An invalid combination.
- A If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the READ statement is executed.

Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

- B The INTO identifier phrase makes a READ statement equivalent to:

```
READ file-name RECORD  
MOVE record-name TO identifier
```

After successful execution of the READ statement, the current record becomes available both in the record-name and identifier.

When the INTO identifier phrase is specified, the current record is moved from the input area to the identifier area according to the rules for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is transferred to identifier.

The INTO identifier phrase cannot be specified when the file contains records of various sizes, as indicated by their record descriptions. The storage area associated with identifier and the record area associated with the file-name cannot be the same storage area.

- D1 When the AT END condition is recognized, a successful CLOSE statement, followed by a successful OPEN statement, must be executed for this file before executing a READ statement.
- D2 When the AT END condition is recognized, a sequential access READ statement for this file must not be executed without first executing one of the following:
- A successful CLOSE statement followed by a successful OPEN statement.
 - A successful START statement for this file.
 - A successful random access READ statement for this file.
- E If no next logical record exists in the file when the READ statement is executed, an AT END condition occurs, and READ statement execution is unsuccessful. The following actions are taken, in the order listed:

1. If the FILE STATUS clause is specified, the status key is updated to indicate an AT END condition.
2. If the AT END phrase is specified, control is transferred to the AT END imperative-statement. Any EXCEPTION/ERROR procedure for this file is not executed.
3. If the AT END phrase is not specified, any EXCEPTION/ERROR procedure for this file is executed.

The AT END phrase must be specified if no explicit or implicit EXCEPTION/ERROR procedure is specified for this file.

IBM Extension

- F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

The literal or identifier must be a character-string of 10 characters or less. If an identifier is specified, it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

A value of all blanks is treated as though the FORMAT phrase was not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

End of IBM Extension

- G1 The record that is made available by the READ statement is determined as follows:
- If the current record pointer was set by the execution of an OPEN statement, the record pointed to is made available.
 - If the current record pointer was set by the execution of a previous READ statement, the pointer is updated to point to the next existing record in the file. That record is then made available.
- G2 The record that is made available by the READ statement is determined as follows:
- If the current record pointer was set by the execution of a START or OPEN statement, the record pointed to is made available if it is still accessible through the path indicated by the current record pointer. If the record is no longer accessible

(due, for example, to deletion of the record), the current record pointer is updated to indicate the next existing record in the file. That record is then made available.

- If the current record pointer was set by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file. That record is then made available.

If the RELATIVE KEY phrase is specified for this file, READ statement execution updates the RELATIVE KEY data item to indicate the relative record number of the record being made available.

G3 The record with the relative record number contained in the RELATIVE KEY data item is made available. If the file does not contain such a record, the INVALID KEY condition exists, and READ statement execution is unsuccessful.

G4 The record made available by the READ statement is determined as follows:

- If the current record pointer was set by the execution of a START or OPEN statement, the record pointed to is made available if it is still accessible through the path indicated by the current record pointer. If the record is no longer accessible (due, for example, to deletion of the record), the current record pointer is updated to indicate the next existing record in the file. That record is then made available.
- If the current record pointer was set by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file. That record is then made available.

IBM Extension

For a file that allows duplicate keys (the DUPLICATES phrase is specified in the file-control entry), the records with duplicate key values are made available in the order specified when the file was created. The system options are first-in first-out (FIFO) and last-in first-out (LIFO). See the LIFO keyword in the *CPF Reference Manual – DDS*.

End of IBM Extension

G5 The record in the file with a key value equal to that of the RECORD KEY data item is then made available. If the file does not contain such a record, the INVALID KEY condition exists, and READ statement execution is unsuccessful.

For a file that allows duplicate keys (the *DUPLICATES* phrase is specified in the file-control entry), the first record with the specified key value is made available. The first record is determined by the order specified when the file was created. The system options are first-in first-out (FIFO) and last-in first-out (LIFO). See the LIFO keyword in the *CPF Reference Manual- DDS*.

End of IBM Extension

- I Allowed when the file is opened for INPUT.
- N If *SELECT OPTIONAL* is specified in the file-control entry for this file and the file is not available when this program runs, execution of the first *READ* statement causes an *AT END* condition. Since the file is not available, the standard system end-of-file processing is not done when the file is closed.
- O Optional.
- P Allowed when the file is opened for I-O.
- U The *INVALID KEY* phrase must be specified for files in which an appropriate *USE* procedure is not specified.
- V If end of volume is recognized during execution of a *READ* statement and logical end of file has not been reached, the following actions are taken in the order listed:
1. The standard ending volume label procedure is executed.
 2. A volume switch occurs.
 3. The standard beginning volume label procedure is executed.
 4. The first data record of the next volume is made available.
- The program receives no indication that the above actions occurred during the read operation.
- W If specified, the key as defined for the specified format is used to get a record of that format. If a record of that format is not found, a record-not-found condition is returned. This occurs even when there are records that have the defined key, but that have a different record format.

If the format is omitted, the common key for the file is used to get the first record of any format that has that common key value. The common key for a file consists of the key fields common to all formats of a file for records residing on the data base. The common key for a file is the leftmost key fields that are common across all

record formats in the file. The common key is built from the data in the record description area using the first record format defined in the program for the file.

X If specified, the next record in the keyed sequence access path that has the requested format is made available. If omitted, the next record in the keyed sequence access path is made available.

Y

	FORMAT Phrase	
	Specified	Omitted
NEXT	Y1	Y2
PRIOR	Y3	Y4
FIRST	Y5	Y6
LAST	Y7	Y8
None of the Above	Y9	Y10

Y1 The next record in the keyed sequence access path having the specified format is made available.

Y2 The next record in the keyed sequence access path is made available regardless of its format.

Y3 The record in the keyed sequence access path preceding the record identified by the current record pointer having the specified format is made available.

Y4 The record in the keyed sequence access path preceding the record identified by the current record pointer is made available regardless of its format.

Y5 The first record in the keyed sequence access path having the specified format is made available.

Y6 The first record in the keyed sequence access path is made available regardless of its format.

Y7 The last record in the keyed sequence access path having the specified format is made available.

Y8 The last record in the keyed sequence access path is made available regardless of its format.

Y9 The key as defined for the specified format is used to get a record of that format. If a record of that format is not found, a record-not-found condition is returned. This occurs even when there are records that have the defined key, but that have a different record format.

Y10 The common key for the file is used to get the first record of any format that has that common key value. The common key for a file consists of the key fields common to all formats of a file for records residing on the data base. The common key for a file consists of the leftmost key fields that are common across all record formats in the file. The common key is built from the data in the record description area using the first record format defined in the program for the file.

Z1 When specified, a sequential read is done (see G2). When omitted, a random read is done (see G3).

Z2 When specified, a sequential read is done (see G4). When omitted, a random access read is done (see G5).

Z3 When specified, a sequential read is done (see G4). If NEXT, FIRST, LAST and PRIOR are all omitted, a random access read is done (see G5).

REWRITE Statement

The REWRITE statement logically replaces an existing record in a file. When the REWRITE statement is executed, the associated file must be open. The formats for the REWRITE statement are as follows:

Format 1

```
REWRITE record-name [FROM identifier-1]
[FORMAT IS {identifier-2}
             {literal-1}]
[INVALID KEY imperative-statement]
```

Format 2 – TRANSACTION

For a discussion of this format, see Chapter 7.

Record-name:

- Must be the name of a record in the File Section
- Must have the same number of character positions as the record being replaced
- Must not be subscripted or indexed
- Can be qualified.

After successful execution of a REWRITE statement, the logical record is no longer available in record-name unless the associated file is named in a

SAME RECORD AREA clause. In this case, the record is also available as a record of the files named in the SAME RECORD AREA clause.

The current record pointer is not affected by execution of the REWRITE statement.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the REWRITE statement is executed.

The following tables illustrate organization, access, and device considerations for the REWRITE statement. The letter codes used in the tables are defined in the section following the tables.

Sequential Organization

Access	SEQUENTIAL										
	Device	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	TAPEFILE	DISKETTE	DISK	DATABASE	FORMATFILE
REWRITE Verb	—	—	—	—	—	—	—	—	A, P	A, P	—
FROM	—	—	—	—	—	—	—	—	O, B	O, B	—
INVALID KEY	—	—	—	—	—	—	—	—	—	—	—
FORMAT	—	—	—	—	—	—	—	—	—	—	—

Relative Organization

Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM
REWRITE Verb	A, P, Z	S, P, Z	S, P, Z	A, P, Z	S, P, Z	S, P, Z
FROM	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	—	U, H, J	U, H, J	—	U, H, J	U, H, J
FORMAT	—	—	—	—	—	—

Indexed Organization

Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM
REWRITE Verb	A, E, P, Z	D, P, Z	D, P, Z	A, E, P, Z	D, P, Z	D, P, Z
FROM	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	U, G, J	U, H, J	U, H, J	U, G, J	U, H, J	U, H, J
FORMAT	—	—	—	—	M, F	M, F

**Letter
Code Meaning**

- An invalid combination.
- A The last input/output statement executed for this file must have been a successfully executed READ statement. When the REWRITE statement is executed, the record retrieved by that READ statement is replaced.
- B The FROM identifier phrase makes a REWRITE statement equivalent to:
- MOVE identifier TO record-name
REWRITE record-name.
- After successful execution of the REWRITE statement, the current record is no longer available in record-name, but is still available in identifier. Record-name and identifier cannot both refer to the same storage area.
- D The record to be replaced is specified by the value in the RECORD KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

IBM Extension

When EXTERNALLY-DESCRIBED-KEY is specified for the file, the key field in the record area for the format specified by the FORMAT phrase (or, if not specified, the first format defined in the program for the file), is used to find the record to be replaced.

If the DUPLICATES phrase was specified for this file, the last input/output statement executed for this file before the REWRITE statement must have been a successfully executed READ statement. The record read by that statement is the one that is replaced. In this case, the FORMAT phrase is not used in determining the record to be replaced.

The READ statement is required to ensure that the proper record is replaced when there are duplicates. If a successful read operation did not occur before the rewrite operation:

- The file status key, if defined, is set to 94.
- The EXCEPTION/ERROR procedure, if any, is executed.
- The REWRITE statement is not executed.

Programming Note: The only way to rewrite one of a sequence of records with duplicate keys is to sequentially read each of the duplicate records and rewrite the desired one.

End of IBM Extension

E The value of the RECORD KEY data item must not have been changed since the record was read.

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

The literal or identifier must be a character-string of 10 characters or less. If an identifier is specified, it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

End of IBM Extension

G Executed when the value contained in the RECORD KEY of the record to be replaced does not equal the RECORD KEY data item of the last retrieved record from the file.

H Executed when the record specified by the key field in the record area is not found.

J When an INVALID KEY condition exists, the updating operation does not take place. The data in record-name is unaffected.

M Optional when processing a file that has one record format.

O Optional.

P Allowed when the file is opened for I-O.

S The record to be replaced is specified by the value in the RELATIVE KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

U The INVALID KEY phrase must be specified for files in which an applicable USE procedure is not specified.

IBM Extension

Z The action of this statement can be inhibited at program execution time by the inhibit write (INHVRT) parameter of the Override with Data Base File (OVRDBF) CL command. When this parameter is specified, non-zero file status codes are not set for data dependent errors. Duplicate key and data conversion errors are examples of data dependent errors.

See the *CL Reference Manual* for more information on this command.

End of IBM Extension

ROLLBACK Statement

The ROLLBACK statement provides a way to cancel one or more changes to data base records when the changes should not remain permanent. The format of the ROLLBACK statement is:

Format

ROLLBACK

When the ROLLBACK statement is executed, any changes made to files under commitment control since the last commitment boundary are removed from the data base.⁸ A commitment boundary is the previous occurrence of a ROLLBACK or COMMIT statement. If no COMMIT or ROLLBACK has been issued, the commitment boundary is the first OPEN of a file under commitment control. Removal of changes takes place for all files under commitment control in the job, and not just for files under commitment control in the COBOL program that issues the ROLLBACK.

Once the ROLLBACK is successfully executed, all record locks held by the job for files under commitment control are released and the records become available to other jobs.

The ROLLBACK has no effect on files not under commitment control. If a ROLLBACK is executed and there are no files under commitment control, the ROLLBACK is ignored.

⁸ When a file is cleared while being opened for OUTPUT, execution of a ROLLBACK statement does not restore cleared records to the file.

A file under commitment control can be opened or closed without affecting the status of changes made since the last commitment boundary. A COMMIT must still be issued to make the changes permanent. A ROLLBACK, when executed, leaves files in the same open or closed state as before execution.

The ROLLBACK statement does *not*:

- modify the I-O-FEEDBACK area for any file
- set a file status value for any file.

For the ROLLBACK statement, the following considerations apply:

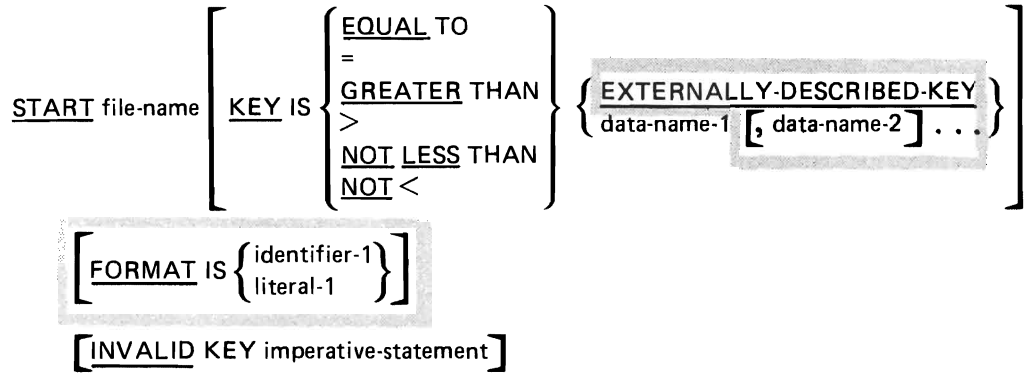
- The ROLLBACK statement sets the current record pointer to the pointer's position at the previous commitment boundary. This is important to remember if you are doing sequential processing.
- If no COMMIT statement has been issued since the file was opened, the ROLLBACK statement sets the current record pointer to the pointer's position at the OPEN.
- The current record pointer is undefined after a ROLLBACK if the file is closed with uncommitted changes.

At the end of every job, an implicit ROLLBACK of uncommitted records is automatically done for all files under commitment control. Any uncommitted changes to the data base are canceled.

START Statement

The START statement provides a way of positioning within an indexed or relative file for subsequent sequential retrieval. This positioning is achieved by comparing the key values of records in the file with the value you place in the RECORD KEY portion of a file's record area (for an indexed file), or in the RELATIVE KEY data item (for a relative file) prior to execution of the START statement. The format for the START statement is as follows:

Format



File-name must be defined in an FD entry in the Data Division. It must not name a sort or merge file.

Data-name-1, data-name-2... can be qualified.

If the FILE STATUS clause is specified in the FILE-CONTROL paragraph, the associated status key is updated when the START statement is executed.

Programming Note: If a START operation with a KEY phrase is unsuccessful, the current record pointer is positioned to the first record in the file, and the record is locked. A WRITE operation for an indexed or sequential file does not release a record lock.

The following tables illustrate organization, access, and device considerations for the START statement. The letter codes used in the tables are defined in the section following the tables.

Sequential Organization

	Access	SEQUENTIAL
	Device	Any
START Verb		Not allowed

Relative Organization

Device	DISK			DATABASE		
	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
START Verb	I, P, A	—	I, P, A	I, P, A	—	I, P, A
KEY IS	O, E	—	O, E	O, E	—	O, E
INVALID KEY	O, U, D	—	O, U, D	O, U, D	—	O, U, D
FORMAT	—	—	—	—	—	—

Indexed Organization

Device	DISK			DATABASE		
	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
START Verb	I, P, B	—	I, P, B	I, P, B, K	—	I, P, B, K
KEY IS	O, G	—	O, G	O, G	—	O, G
INVALID KEY	O, U, D	—	O, U, D	O, U, D	—	O, U, D
FORMAT	—	—	—	O, F, J	—	O, F, J

Letter Code Meaning

- An invalid combination.
- A When the KEY phrase is not specified, the current record pointer is set to the record in the file with a key (relative record number) equal to the RELATIVE KEY data item.
- B When the KEY phrase is not specified, the current record pointer is set to the record with a key equal to the value contained in the RECORD KEY data item.
- D If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists. The position of the current record pointer is undefined, and the INVALID KEY imperative-statement, if specified, is executed.
- E When the KEY phrase is specified, data-name-1 must specify the RELATIVE KEY. The current record pointer is positioned to the first logical record currently existing in the file with a key (relative record number) that satisfies the comparison with the RELATIVE KEY data item.

IBM Extension

- F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

The literal or identifier must be a character-string of 10 characters or less. If an identifier is specified, it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

End of IBM Extension

- G When the KEY phrase is specified, the search argument used for the comparison is data-name-1, which can be:

- The RECORD KEY itself.
- An alphanumeric data item within a record description for the file with a leftmost character position that corresponds to the leftmost character position of the key field in the record area. This data item must be less than or equal to the length of the RECORD KEY for the file. This data item can be qualified.

Note: If the RECORD KEY is defined as COMP, COMP-3, or COMP-4, the key data item must be the RECORD KEY itself. A partial key field in the record area cannot be used.

The current record pointer is positioned to the first record in the file with a record key for a format that satisfies the comparison. If the operands in the comparison are of unequal length, the comparison proceeds as if the longer field were truncated on the right to the length of the shorter field. All other numeric and nonnumeric comparison rules apply, except that the PROGRAM COLLATING SEQUENCE, if specified, has no effect.

IBM Extension

For a file that specified RECORD KEY IS EXTERNALLY-DESCRIBED-KEY, the following additional considerations apply:

- The reserved word EXTERNALLY-DESCRIBED-KEY can be specified. This indicates that the complete key field in the record area should be used in the comparison.
- A series of data names can be specified. This allows a partial key field in the record area to be used (generic START). These data names must follow the following rules:
 - All except the last of the data names specified must be a record key for a format that was copied in for the file. The record format in which they are contained does not have to be the one that can be specified by the FORMAT phrase.
 - The order of these data names (key fields) must match the order of the keys as defined in DDS; that is, they must be specified from most significant field to least significant.
 - The total number of data names cannot exceed the number of key fields defined for that record format.
 - If the last data name specified in the series is not a key field in the record area, it must have its left byte occupy the same space as the key field that is defined at that relative position. If the key field in the record area at this position is a COMP, COMP-3, or COMP-4 field, only the key field itself can be used as the data name.
- The following table shows the action between the KEY IS phrase and the FORMAT phrase:

		KEY Phrase	
FORMAT Phrase	Data-name Series	Omitted	EXTERNALLY-DESCRIBED-KEY
Yes	G1, G2	G3, G4	G3, G2
No	G1, G5	G6, G7	G6, G5

- G1 The search argument is built using the specified data items.
- G2 The current record pointer is set to the first record in the file of the format specified with a record key that satisfies the comparison specified in the key phrase.
- G3 The search argument is built using the key fields in the record area for the format specified in the FORMAT phrase.
- G4 The current record pointer is set to the first record in the file of the specified format with a record key equal to the search argument.

- G5 The current record pointer is set to the first record in the file with a common key for the file that satisfies the comparison specified in the KEY phrase. If there is no common key, the current record pointer is set to the first record in the file.
- G6 The search argument is built using the key fields in the record area for the first record format for the file as defined in the program.
- G7 The current record pointer is set to the first record in the file with a common key for the file that is equal to the search argument. If there is no common key, the current record pointer is set to the first record in the file.

End of IBM Extension

- I Allowed when the file is opened for INPUT.
- J If specified, the current record pointer is set to the first record of the specified record format that satisfies the comparison. If omitted, the current record pointer is set to the first record of any format that satisfies the comparison.

See the table in G (under "KEY IS Phrase") for a description of how this interacts with EXTERNALLY-DESCRIBED-KEY and the KEY IS phrase.

IBM Extension

- K The meaning of the comparison can be affected by the type of key fields in the record area defined for the file. Key fields on this system can be defined as multiple fields, each of which can be in ascending or descending sequence. The system establishes a sequence (keyed sequence access path) for the records based on the values contained in the record key for the format and the sequencing specified in DDS. When a START statement is executed, the request is interpreted as follows:

COBOL Comparison	System Result
GREATER THAN	AFTER
NOT LESS THAN	EQUAL TO or AFTER

For example, when a statement is executed using the comparison of GREATER THAN, a search is made of these sequenced records for the first record after the search argument specified by the START statement. If the file was sequenced using descending keys, the current record pointer would point to a record with a key less than the one specified and not greater than that specified in the START statement.

End of IBM Extension

- O Optional.
- P Allowed when the file is opened for I-O.
- U The INVALID KEY phrase must be specified for files in which an appropriate USE procedure is not specified.

WRITE Statement

The WRITE statement releases a record to the system. The formats for the WRITE statement are as follows:

Format 1 – Sequential Files

```
WRITE record-name [ FROM identifier-1 ]
    [ { BEFORE } ADVANCING { { identifier-2 } { LINE }
      { AFTER }           { integer } { LINES } } ]
    [ { mnemonic-name }
      { PAGE } ]
    [ AT { END-OF-PAGE } imperative-statement
      { EOP } ]
```

Format 2 – Indexed and Relative Files

```
WRITE record-name [ FROM identifier-1 ]
    [ FORMAT IS { identifier-2 }
      { literal-1 } ]
    [ INVALID KEY imperative-statement ]
```

Format 3 – FORMATFILE Files

```
WRITE record-name [ FROM identifier-1 ]
    [ FORMAT IS { identifier-2 }
      { literal-1 } ]
    [ { INDICATOR [ IS ] identifier-3
      { INDICATORS [ ARE ]
      { INDIC } ] ]
    [ AT { END-OF-PAGE } imperative-statement
      { EOP } ]
```

Format 4 – TRANSACTION File (Nonsubfile)

Format 5 – TRANSACTION File (Subfile)

See Chapter 7 for a discussion of these formats.

Record-name:

- Must be the name of a record in the File Section of the Data Division
- Can be qualified
- Cannot be associated with a sort or merge file.

The maximum record size for a data base file is established at the time the file is defined to the system (using the Create Physical File [CRTPF] or the Create Logical File [CRTLF] CL command) and cannot be changed. If the record length defined in the program is incompatible with the record length defined to the system, the following occurs during output to the file:

- When the program record length is greater than the length defined to the system, the records are truncated to the system length. If the file is empty, the program record length is used.
- When the program record length is less than the record length defined in the system, the records are padded with blanks to make them the size specified in the system.

Execution of the WRITE statement releases a record to the file associated with record-name. After execution of a WRITE statement, the record is no longer available in record-name unless either of the following is true:

- The associated file is named in a SAME RECORD AREA clause. In this case, the record is also available as a record of the files named in the SAME RECORD AREA clause.
- The WRITE statement is unsuccessful due to a boundary violation (beyond extent).

If either of the above conditions is true, the record is still available in record-name.

The current record pointer is not affected by execution of the WRITE statement.

The number of character positions required to store the record in a file can be, but is not necessarily, the same as the number of character positions defined by the description of the record in the COBOL program. (See “PICTURE Clause” and “Usage Clause” in Chapter 4.)

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the WRITE statement is executed.

When an attempt is made to write beyond the externally defined boundaries of the file, WRITE statement execution is unsuccessful, and an

EXCEPTION/ERROR condition exists. The status key, if specified, is updated. If an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure is executed. If no such procedure is specified, the results are unpredictable.

The following tables illustrate organization, access, and device considerations for the WRITE statement. The letter codes used in the tables are defined in the section following the tables.

Sequential Organization

Access	SEQUENTIAL										
	Device	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	TAPEFILE	DISKETTE	DISK	DATABASE	FORMATFILE
WRITE Verb	—	Q, T	Q, T	Q, T	Q, T	Q, T	V, Q, S, T	V, Q, T	Q, S, T, Z	Q, S, T, Z	Q, T
FROM	—	O, B	O, B	O, B	O, B	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	—	—	—	—	—	—	—	—	—	—	—
ADVANCING	—	O, D2	O, D2	O, D2	O, D1	—	—	—	—	—	—
AT END-OF-PAGE	—	—	—	—	O, E1	—	—	—	—	—	O, E2
FORMAT	—	—	—	—	—	—	—	—	—	—	N, F
INDICATORS	—	—	—	—	—	—	—	—	—	—	I

Relative Organization

Device	DISK			DATABASE			
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
WRITE Verb	Q, K, Z	P, Q, M, Z	P, Q, M, Z	P, Q, M, Z	Q, K, Z	P, Q, M, Z	P, Q, M, Z
FROM	O, B	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	O, J1, J, U	O, J1, J2, J, U	O, J1, J2, J, U	O, J1, J2, J, U	O, J1, J, U	O, J1, J2, J, U	O, J1, J2, J, U
ADVANCING	—	—	—	—	—	—	—
AT END-OF-PAGE	—	—	—	—	—	—	—
FORMAT	—	—	—	—	—	—	—
INDICATORS	—	—	—	—	—	—	—

Indexed Organization

Device	DISK			DATABASE		
	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
WRITE Verb	G, H1, Q, Z	G, H2, P, Q, Z	G, H2, P, Q, Z	G, H1, Q, Z	G, H2, P, Q, Z	G, H2, P, Q, Z
FROM	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	O, J1, J4, J, U	O, J1, J3, J, U	O, J1, J3, J, U	O, J1, J4, J, U	O, J1, J3, J, U	O, J1, J3, J, U
ADVANCING	—	—	—	—	—	—
AT END-OF-PAGE	—	—	—	—	—	—
FORMAT	—	—	—	N, F	N, F	N, F
INDICATORS	—	—	—	—	—	—

Letter Code Meaning

— An invalid combination.

A When the KEY phrase is not specified, the current record pointer is set to the record in the file with a key (relative record number) equal to the RELATIVE KEY data item.

B The FROM identifier phrase makes a WRITE statement equivalent to:

MOVE identifier TO record-name
WRITE record-name.

After successful execution of the WRITE statement, the current record pointer is no longer available in record-name, but is still available in identifier. Record-name and identifier cannot both refer to the same storage area.

D1 When not specified, a default of AFTER ADVANCING 1 LINE is used. When specified, the following rules apply:

- When BEFORE ADVANCING is specified, the line is printed before the page advances.
- When AFTER ADVANCING is specified, the page advances before the line is printed.
- When identifier-2 is specified, the page advances the number of lines equal to the current value in identifier-2. Identifier-2 must name an elementary integer data item. Identifier-2 can be zero.
- When integer is specified, the page advances the number of lines equal to the value of integer. Integer can be zero.
- When a mnemonic-name is specified, a page eject or space suppression occurs. The mnemonic-name must be equated with

function-name-1 in the SPECIAL-NAMES paragraph. This phrase is not valid if a LINAGE clause is specified in the FD entry for this file.

- When PAGE is specified, the record is printed on the logical page BEFORE or AFTER (depending on the phrase specified) the device is positioned to the next logical page. If PAGE has no meaning for the device used, BEFORE or AFTER ADVANCING 1 LINE is provided (depending on the phrase specified).

If the FD entry contains a LINAGE clause, the device is positioned to the first printable line of the next page, as specified in that clause. If the LINAGE clause is omitted, the device is positioned to line 1 of the next page.

If the LINAGE clause is specified for this file, the associated LINAGE-COUNTER special register is modified during the execution of the WRITE statement, according to the following rules:

- If ADVANCING PAGE is specified, LINAGE-COUNTER is reset to 1.
- If ADVANCING identifier-2 or integer is specified, LINAGE-COUNTER is incremented by the value of identifier-2 or integer.
- If the ADVANCING phrase is omitted, LINAGE-COUNTER is incremented by 1.
- When the device is repositioned to the first printable line of a new page, LINAGE-COUNTER is reset to 1.

D2 When not specified, stacker 1 is selected.

IBM Extension

The ADVANCING phrase using a mnemonic name can be specified to control the stacker selection. The mnemonic name must be equated with function-name-1 in the SPECIAL-NAMES paragraph and must be one of the valid stacker selection names.

End of IBM Extension

E1 The keywords END-OF-PAGE and EOP are equivalent. When the END-OF-PAGE phrase is specified, the FD entry for this file must contain a LINAGE clause. When END-OF-PAGE is specified, and the logical end of the printed page is reached during execution of the WRITE statement, the END-OF-PAGE imperative-statement is executed. The logical end of the printed page is specified in the LINAGE clause associated with record-name.

An END-OF-PAGE condition is reached when execution of a WRITE END-OF-PAGE statement causes printing or spacing within the footing area of a page body. This occurs when execution of such a WRITE statement causes the value in the LINAGE-COUNTER to equal

or exceed the value specified in the WITH FOOTING phrase of the LINAGE clause. The WRITE statement is executed, and then the END-OF-PAGE imperative statement is executed.

An automatic page overflow condition is reached whenever the execution of any WRITE statement with or without the END-OF-PAGE phrase cannot be completely executed within the current page body. This occurs when an executed WRITE statement would cause the value in the LINAGE-COUNTER to exceed the number of lines for the page body specified in the LINAGE clause. In this case, the line is printed before or after the device is repositioned to the first printable line on the next logical page, as specified in the LINAGE clause.

If the END-OF-PAGE phrase is specified, the END-OF-PAGE imperative-statement is then executed. The END-OF-PAGE condition and automatic page overflow condition occur simultaneously in the following cases:

- When the WITH FOOTING phrase of the LINAGE clause is not specified. This results in no distinction between the END-OF-PAGE condition and the page overflow condition. No footing information can be printed at the bottom of a logical page when the FOOTING phrase is not specified.
- When the WITH FOOTING phrase is specified, but the execution of a WRITE statement would cause the LINAGE-COUNTER to exceed both the footing value and the page body value specified in the LINAGE clause.

E2 The keywords END-OF-PAGE and EOP are equivalent. When the END-OF-PAGE is specified, and the logical end of page is reached, during execution of the WRITE statement for the FORMATFILE file, the END-OF-PAGE imperative statement is executed. The logical end of the printed page is specified in the overflow line number parameter of the CRTPRTF command or the OVRPRTF command.

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

The literal or identifier must be a character-string of 10 characters or less. If an identifier is specified, it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

End of IBM Extension

- G When the WRITE statement is executed, the system releases the record. Before the WRITE statement is executed, the user must set the key fields in the record area to the desired value.

IBM Extension

If the DUPLICATES phrase is specified, record key values for a format need not be unique (see "FILE-CONTROL Paragraph, RECORD KEY Clause [Indexed File]" in Chapter 3.) In this case, the system stores the records so that later sequential access to the records allows retrieval in the order specified in DDS.

End of IBM Extension

- H1 Records must be released in ascending RECORD KEY value sequence.

Note: The records must be released in ascending key sequence even though the file can be ordered in descending key sequence by a DDS option.

- H2 Records can be released in any user-specified order.

- I See "Indicators" in Chapter 7.

- J When the INVALID KEY condition is recognized, WRITE statement execution is unsuccessful, and the contents of the record are unaffected. See the "RECORD KEY Clause" of the "SELECT Statement" under "FILE-CONTROL Paragraph" in Chapter 3 for the order of the actions taken.

J1 An INVALID KEY condition exists when an attempt is made to write beyond the externally defined boundaries of the file.

J2 An INVALID KEY condition exists when the relative key specifies a record that already contains data.

J3 An INVALID KEY condition exists when the value of the key field in the record area equals that of an already existing record and DUPLICATES are not allowed.

J4 An INVALID KEY condition exists when the value of the key field in the record area is not greater than that for the previous record.

IBM Extension

If DUPLICATES are allowed, this condition exists only if the RECORD KEY is less than that for the previous record.

End of IBM Extension

- K The first record released has relative record number 1, the second has number 2, the third has number 3, and so on. If the RELATIVE KEY is specified in the file-control entry, the relative record number of the record just released is placed in the RELATIVE KEY during execution of the WRITE statement.
- M The RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, this record is placed at the specified relative record number position in the file, if this position is vacant.
- N Required if there is more than one record format for the file.
- O Optional.
- P Allowed when the file is opened for I-O.
- Q Allowed when the file is opened for OUTPUT.
- S Allowed when the file is opened for EXTEND.
- T When an attempt is made to write beyond the externally defined boundaries of the file, the execution of the WRITE statement is unsuccessful and an EXCEPTION/ERROR condition exists. The contents of record-name are unaffected. If specified, the status key is updated, and if an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure is executed. If no such procedure is specified, the results are unpredictable.
- U The INVALID KEY phrase must be specified for files in which an applicable USE procedure is not specified.
- V When end-of-volume is recognized for a multivolume OUTPUT file, the WRITE statement performs the following operations in the following order:
1. The standard ending volume label procedure is executed.
 2. A volume switch occurs.
 3. The standard beginning volume label procedure is executed.
- No indication that an end-of-volume has occurred is returned to the program.

Z The action of this statement can be inhibited at program execution time by the INHWRT parameter of the OVRDBF CL command. When this parameter is specified, non-zero file status codes are not set for data dependent errors. Duplicate key and data conversion errors are examples of data dependent errors.

See the *CL Reference Manual* for more information on this command.

Arithmetic Statements

Arithmetic statements are used for computations. Individual operations are specified by the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. The COMPUTE statement can be used to symbolically combine these operations in a formula.

Arithmetic Statement Operands

The data description of operands in an arithmetic statement need not be the same. Throughout the calculation, the compiler supplies any necessary data conversion and decimal point alignment.

Size of Operands

The maximum size of each operand is 18 decimal digits. The composite of operands (a hypothetical data item resulting from the superimposition of the operands aligned by an assumed decimal point) must not contain more than 18 decimal digits.

For the ADD and SUBTRACT statements, the composite of operands is determined by superimposing all operands in a given statement except those following the word GIVING.

For the MULTIPLY statement, the composite of operands is determined by superimposing all receiving data items.

For the DIVIDE statement, the composite of operands is determined by superimposing all receiving data items except the REMAINDER data item.

For the COMPUTE statement, the restriction on composite of operands does not apply.

For example, the items A, B, and C are defined in the Data Division as follows:

```
01 A PICTURE S9(7)V9(5).
01 B PICTURE S9(11)V99.
01 C PICTURE S9(12)V9(3)
```

If the statement `ADD A, B TO C` is executed, then the composite of operands for this statement consists of 17 decimal digits. It has the following implicit PICTURE clause:

```
PICTURE S9(12)V9(5)
```

IBM Extension

The composite of all operands in an arithmetic statement can have a maximum length of 30 digits.

End of IBM Extension

Overlapping Operands

When operands in an arithmetic statement share part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

Multiple Results

When an arithmetic statement has multiple results, execution conceptually proceeds as follows:

- The statement performs all arithmetic operations to find the result to be placed in the receiving items and stores that result in a temporary location.
- A sequence of statements transfers or combines the value of this temporary result with each single receiving field. The statements are considered to be written in the same left-to-right order that the multiple results are listed.

For example, executing the following statement:

```
ADD A, B, C TO C, D(C), E.
```

is equivalent to executing the following series of statements:

```
ADD A, B, C GIVING TEMP.
ADD TEMP TO C
ADD TEMP TO D(C)
ADD TEMP TO E
```

TEMP is a compiler-supplied temporary result field. When the addition operation for `D(C)` is performed, the subscript `C` contains the new value of `C`.

Note: The compiler does not generate a temporary result field when only one identifier is specified in the following cases: in the `ADD` statement, Format 1, preceding the keyword `TO`; in the `SUBTRACT` statement, Format 1,

preceding the keyword FROM; and in the MULTIPLY and DIVIDE statements, Format 1.

Programming Note: In all arithmetic statements, it is the user's responsibility to define data with enough digits and decimal places to ensure accuracy in the final result. Refer to Appendix E for more information.

Common Phrases

There are several phrases common to the arithmetic statements. They are the CORRESPONDING phrase, the GIVING phrase, the ROUNDED phrase, and the SIZE ERROR phrase. Their description precedes the descriptions of the individual statements.

CORRESPONDING Phrase

The CORRESPONDING phrase allows operations to be performed on elementary items of the same name simply by specifying the group items to which they belong.

The CORRESPONDING phrase is valid in the ADD, SUBTRACT, and MOVE statements. The abbreviation CORR is equivalent to the keyword CORRESPONDING.

Both identifiers following the keyword CORRESPONDING must name group items. In this discussion, these identifiers are referred to as d1 and d2.

A pair of subordinate data items, one from d1 and one from d2, correspond if the following conditions are true:

- In an ADD or SUBTRACT statement, both of the subordinate items are elementary numeric data-items.
- In a MOVE statement, at least one of the subordinate items is elementary.
- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.
- The subordinate items are not identified by the keyword FILLER.
- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group item, the items subordinate to it are also ignored. However, d1 and d2 themselves can contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.

For example, two data hierarchies are defined as follows:

```

05 ITEM-1 OCCURS 6 INDEXED BY X
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C REDEFINES ITEM-B
05 ITEM-2
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C

```

If ADD CORR ITEM-2 TO ITEM-1(X) is specified, ITEM-A and ITEM-A(X) and ITEM-B and ITEM-B(X) are considered to be corresponding and are added together. ITEM-C and ITEM-C(X) are not included because ITEM-C(X) includes a REDEFINES clause in its data description. ITEM-1 is valid as either d1 or d2.

- Neither d1 nor d2 is described as a level 66, 77 or 88 item, or as a FILLER or USAGE IS INDEX item.

IBM Extension

D1 and/or d2 can be subordinate to a FILLER item.

End of IBM Extension

GIVING Phrase

If the GIVING phrase is specified, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it can be a numeric edited item.

ROUNDED Phrase

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

If the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless the ROUNDED phrase is specified. When the ROUNDED phrase is specified, the least significant digit of the resultant identifier has its absolute value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Phrase

A size error condition exists if, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the resultant field. Division by zero or zero raised to the zero power always causes a size error condition.

In the ADD, SUBTRACT, and COMPUTE statements, the size error condition applies only to final results. In the MULTIPLY and DIVIDE statements, the size error condition applies both to final results and to intermediate results.

If the ROUNDED phrase is specified, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the SIZE ERROR phrase is specified.

If the SIZE ERROR phrase is not specified and a size error condition occurs, the value of the affected resultant identifier is unpredictable. When multiple receivers are specified, those that do not have a size error are not affected by receivers that do have the error.

If the SIZE ERROR phrase is specified and a size error condition occurs, the error results are not placed in the receiving identifier. After completion of the execution of the arithmetic operation, the imperative-statement in the SIZE ERROR phrase is executed.

If an individual arithmetic operation causes a size error condition for ADD CORRESPONDING and SUBTRACT CORRESPONDING statements, the SIZE ERROR imperative-statement is not executed until all of the individual additions or subtraction have been completed.

ADD Statement

The ADD statement causes two or more numeric operands to be summed and the result to be stored. The formats of the ADD statement are as follows:

Format 1

$$\text{ADD } \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left[\begin{array}{l} \text{, identifier-2} \\ \text{, literal-2} \end{array} \right] \dots \text{TO identifier-m } \left[\text{ROUNDED} \right] \\ \left[\begin{array}{l} \text{, identifier-n } \left[\text{ROUNDED} \right] \end{array} \right] \dots \left[\text{ON SIZE ERROR imperative-statement} \right]$$

Format 2

$$\underline{\text{ADD}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\}, \left\{ \begin{array}{l} \text{identifier-2} \\ \text{literal-2} \end{array} \right\} \left[\begin{array}{l} \text{, identifier-3} \\ \text{, literal-3} \end{array} \right] \dots$$
$$\underline{\text{GIVING}} \text{ identifier-m } \left[\underline{\text{ROUNDED}} \right] \left[\begin{array}{l} \text{, identifier-n } \left[\underline{\text{ROUNDED}} \right] \\ \dots \end{array} \right]$$
$$\left[\underline{\text{ON SIZE ERROR}} \text{ imperative-statement} \right]$$

Format 3

$$\underline{\text{ADD}} \left\{ \begin{array}{l} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{array} \right\} \text{ identifier-1 } \underline{\text{TO}} \text{ identifier-2 } \left[\underline{\text{ROUNDED}} \right]$$
$$\left[\underline{\text{ON SIZE ERROR}} \text{ imperative-statement} \right]$$

In Formats 1 and 2, each identifier, except those following the keyword GIVING must name an elementary numeric item. In Format 2, each identifier following the keyword GIVING must name an elementary numeric or numeric edited item. In Format 3, each identifier must name a group item. In all formats, each literal must be a numeric literal.

In Format 1, all identifiers or literals preceding the keyword TO are added together, and this sum is added to and stored immediately in identifier-m. If specified, the sum is then added to and stored immediately in identifier-n, and so on.

In Format 2, at least two operands must precede the keyword GIVING. The values of these operands are added together, and the sum is stored as the new value of identifier-m, and, if specified, identifier-n, and so on.

If Format 3, elementary data items within identifier-1 are added to and stored in the corresponding elementary items within identifier-2.

For the ROUNDED and SIZE ERROR phrases, and for operand considerations, refer to the preceding "Common Phrases" in this section.

COMPUTE Statement

The COMPUTE statement assigns the value of an arithmetic expression to one or more data items.

Format

COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] . . .
= arithmetic-expression [ON SIZE ERROR imperative-statement]

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on the composite operands and/or receiving data items imposed by the rules for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements.

The identifiers that appear to the left of the equal sign (=) must name either elementary numeric items or elementary numeric edited items.

When the COMPUTE statement is executed, the value of the arithmetic expression is calculated; then this value is stored as the new value of identifier-1, identifier-2, and so on, in turn.

The arithmetic expression can be any meaningful combination of elementary numeric items, numeric literals, and arithmetic operators.

An arithmetic expression consisting of a single identifier or literal allows the user to set identifier-1, and so on, equal to the value of that identifier or literal.

For the ROUNDED and SIZE ERROR phrases, and for operand considerations, see "Common Phrases" earlier in this section.

DIVIDE Statement

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder. The formats of the DIVIDE statement are:

Format 1

DIVIDE { identifier-1
literal-1 } INTO identifier-2 [ROUNDED]
[, identifier-3 [ROUNDED]] . . . [ON SIZE ERROR imperative-statement]

Format 2

$$\underline{\text{DIVIDE}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left\{ \begin{array}{l} \text{INTO} \\ \underline{\text{BY}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-2} \\ \text{literal-2} \end{array} \right\} \underline{\text{GIVING}} \text{identifier-3} \left[\underline{\text{ROUNDED}} \right] \\ \left[, \text{identifier-4} \left[\underline{\text{ROUNDED}} \right] \right] \dots \left[\underline{\text{ON SIZE ERROR}} \text{imperative-statement} \right]$$

Format 3

$$\underline{\text{DIVIDE}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left\{ \begin{array}{l} \text{INTO} \\ \underline{\text{BY}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-2} \\ \text{literal-2} \end{array} \right\} \underline{\text{GIVING}} \text{identifier-3} \left[\underline{\text{ROUNDED}} \right] \\ \underline{\text{REMAINDER}} \text{identifier-4} \left[\underline{\text{ON SIZE ERROR}} \text{imperative-statement} \right]$$

Each identifier except those following the keywords `GIVING` and `REMAINDER` must name an elementary numeric item. Each identifier following the keywords `GIVING` and `REMAINDER` must name an elementary numeric or numeric edited item. Each literal must be a numeric literal.

In Format 1, the value of `literal-1` or `identifier-1` is divided into the value of `identifier-2`; then the quotient is placed in `identifier-2`. If `identifier-3` is specified, the value of `literal-1` or `identifier-1` is divided into `identifier-3`; then the quotient is placed in `identifier-3`, and so on.

In Format 2, the value of `identifier-1` or `literal-1` is divided into/by the value of `identifier-2` or `literal-2`. The value of the quotient is stored in `identifier-3`, and (if specified) `identifier-4`, and so on.

In Format 3, the value of `identifier-1` or `literal-1` is divided into/by `identifier-2` or `literal-2`. The value of the quotient is stored in `identifier-3`, and the value of the remainder is stored in `identifier-4`.

The remainder is defined as the result of subtracting the product of the quotient and the divisor from the dividend. If `identifier-3` (the quotient) is a numeric edited field, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient.

For the `ROUNDED` and `SIZE ERROR` phrases, and for operand considerations, see “Common Phrases” earlier in this section.

In addition to the conditions for common phrases, the following considerations apply when the `ROUNDED` and `SIZE ERROR` phrases are used in Format 3.

- When the `ROUNDED` phrase is specified, the quotient used to calculate the remainder is an intermediate field which contains the quotient truncated rather than rounded.
- When the `ON SIZE ERROR` phrase is specified and the size error condition occurs on the quotient, no remainder calculation is

meaningful. Therefore, the contents of the quotient field (identifier-3) and the remainder field (identifier-4) are unchanged.

- When the ON SIZE ERROR phrase is specified and the size error occurs on the remainder, the contents of the remainder field (identifier-4) are unchanged.

Note: In the two preceding cases, the user must analyze the results to determine which situation has actually occurred.

MULTIPLY Statement

The MULTIPLY statement causes numeric items to be multiplied and sets the values of data items equal to the results. The formats of the MULTIPLY statement are:

Format 1

$$\text{MULTIPLY } \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \text{ BY identifier-2 } \underline{\text{ROUNDED}} \\ \left[, \text{identifier-3 } \underline{\text{ROUNDED}} \right] \dots \left[\text{ON SIZE ERROR imperative-statement} \right]$$

Format 2

$$\text{MULTIPLY } \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \text{ BY } \left\{ \begin{array}{l} \text{identifier-2} \\ \text{literal-2} \end{array} \right\} \underline{\text{GIVING}} \text{ identifier-3 } \underline{\text{ROUNDED}} \\ \left[, \text{identifier-4 } \underline{\text{ROUNDED}} \right] \dots \left[\text{ON SIZE ERROR imperative-statement} \right]$$

Each identifier except those following the keyword GIVING must name an elementary numeric item. Each identifier following the keyword GIVING must name an elementary numeric or numeric edited item. Each literal must be a numeric literal.

In Format 1, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2; the product is then placed in identifier-2. If identifier-3 is specified, the value of identifier-1 or literal-1 is multiplied by the value of identifier-3; the product is then placed in identifier-3, and so on.

In Format 2, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2; the product is then stored in identifier-3, and, if specified, identifier-4, and so on.

For the ROUNDED and SIZE ERROR phrases, and for operand considerations, see "Common Phrases" earlier in this section.

SUBTRACT Statement

The SUBTRACT statement causes either one, or the sum of two or more numeric items to be subtracted from one or more numeric items and the result to be stored. The formats of the SUBTRACT statement are:

Format 1

$$\text{SUBTRACT } \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left[\begin{array}{l} \text{, identifier-2} \\ \text{, literal-2} \end{array} \right] \dots \text{FROM identifier-3 } \underline{\text{ROUNDED}} \\ \left[\text{, identifier-4 } \underline{\text{ROUNDED}} \right] \dots \left[\text{ON SIZE ERROR imperative-statement} \right]$$

Format 2

$$\text{SUBTRACT } \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left[\begin{array}{l} \text{, identifier-2} \\ \text{, literal-2} \end{array} \right] \dots \text{FROM } \left\{ \begin{array}{l} \text{identifier-3} \\ \text{literal-3} \end{array} \right\} \\ \underline{\text{GIVING}} \text{ identifier-4 } \underline{\text{ROUNDED}} \left[\text{, identifier-5 } \underline{\text{ROUNDED}} \right] \dots \\ \left[\text{ON SIZE ERROR imperative-statement} \right]$$

Format 3

$$\text{SUBTRACT } \left\{ \begin{array}{l} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{array} \right\} \text{ identifier-1 FROM identifier-2 } \underline{\text{ROUNDED}} \\ \left[\text{ON SIZE ERROR imperative-statement} \right]$$

In Formats 1 and 2, each identifier except those following the keyword GIVING must name an elementary numeric item. In Format 2, each identifier following the keyword GIVING must name a numeric elementary or numeric edited elementary item. In Format 3, each identifier must name a group item. In all formats, each literal must be a numeric literal.

In Format 1, all identifiers or literals preceding the keyword FROM are added together, and this sum is subtracted from and stored immediately in identifier-3, and then, if specified, subtracted from and stored immediately in identifier-4, and so on.

In Format 2, all identifiers or literals preceding the keyword FROM are added together and this sum is subtracted from identifier-3 or literal-3. The result of the subtraction is stored as the new value of identifier-4, and, if specified, identifier-5, and so on.

In Format 3, elementary data items within identifier-1 are subtracted from and stored in the corresponding elementary data items within identifier-2.

For the `ROUNDED` and `SIZE ERROR` phrases, and for operand considerations, see “Common Phrases” earlier in this section.

Data Manipulation Statements

Movement and inspection of data are the functions of the following COBOL statements: `INSPECT`, `MOVE`, `STRING`, and `UNSTRING`.

When the sending and receiving fields of a data manipulation statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

INSPECT Statement

The `INSPECT` statement specifies that characters in a data item are to be counted, replaced, or counted and replaced. The formats of the `INSPECT` statement are:

Format 1

INSPECT identifier-1 TALLYING

$$\left\{ , \text{identifier-2 } \underline{\text{FOR}} \left\{ , \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \\ \underline{\text{CHARACTERS}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{literal-1} \end{array} \right\} \right. \right. \\ \left. \left. \left[\left\{ \begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array} \right\} \text{INITIAL } \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \right] \right\} \dots \right\} \dots$$

Format 2

INSPECT identifier-1 REPLACING

$$\left\{ \left[\underline{\text{CHARACTERS}} \underline{\text{BY}} \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \left[\left\{ \begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array} \right\} \text{INITIAL } \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\} \right] \right. \right. \\ \left. \left. \left\{ , \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \\ \underline{\text{FIRST}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-5} \\ \text{literal-3} \end{array} \right\} \underline{\text{BY}} \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \left[\left\{ \begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array} \right\} \text{INITIAL } \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\} \right] \right\} \dots \right\} \dots \right\}$$

Format 3

INSPECT identifier-1 TALLYING

$$\left\{ , \text{identifier-2 } \underline{\text{FOR}} \left\{ , \left\{ \left\{ \underline{\text{ALL}} \right\} \left\{ \text{identifier-3} \right\} \right\} \left\{ \left\{ \underline{\text{LEADING}} \right\} \left\{ \text{literal-1} \right\} \right\} \right\} \left\{ \underline{\text{CHARACTERS}} \right\} \right\} \left[\left\{ \left\{ \underline{\text{BEFORE}} \right\} \right\} \text{INITIAL} \left\{ \text{identifier-4} \right\} \right] \left[\left\{ \left\{ \underline{\text{AFTER}} \right\} \right\} \text{INITIAL} \left\{ \text{literal-2} \right\} \right] \right] \dots \dots \dots \left. \right\} \dots \dots$$

REPLACING

$$\left(\left[\underline{\text{CHARACTERS}} \text{ BY } \left\{ \text{identifier-6} \right\} \left[\left\{ \left\{ \underline{\text{BEFORE}} \right\} \right\} \text{INITIAL} \left\{ \text{identifier-7} \right\} \right] \left[\left\{ \left\{ \underline{\text{AFTER}} \right\} \right\} \text{INITIAL} \left\{ \text{literal-5} \right\} \right] \right] \right\} \left\{ , \left\{ \left\{ \underline{\text{ALL}} \right\} \right\} \left\{ \left\{ \underline{\text{LEADING}} \right\} \right\} \left\{ \left\{ \underline{\text{FIRST}} \right\} \right\} \right\} \left\{ , \left\{ \text{identifier-5} \right\} \right\} \text{BY} \left\{ \text{identifier-6} \right\} \left\{ \text{literal-3} \right\} \right\} \left[\left\{ \left\{ \underline{\text{BEFORE}} \right\} \right\} \text{INITIAL} \left\{ \text{identifier-7} \right\} \right] \left[\left\{ \left\{ \underline{\text{AFTER}} \right\} \right\} \text{INITIAL} \left\{ \text{literal-5} \right\} \right] \right] \dots \dots \dots \right. \left. \right)$$

Either the TALLYING or the REPLACING phrase must be specified. Both the TALLYING and REPLACING phrases can be specified. If both TALLYING and REPLACING are specified (Format 3), all tallying is performed before any replacement is made.

Identifier-1 is the inspected item. Identifier-1 must be an elementary or group item with USAGE DISPLAY.

All other identifiers except identifier-2 (the count field) must be elementary alphabetic, alphanumeric, or zoned decimal items. Each is treated according to its data category. Each data category is treated as follows:

- Alphabetic or alphanumeric items are treated as a character-string.
- Alphanumeric edited, numeric edited, or unsigned numeric (zoned decimal) items are treated as though redefined as alphanumeric, and the INSPECT statement refers to the alphanumeric item.
- Signed numeric (zoned decimal) items are treated as though moved to an unsigned zoned decimal item of the same length, and then treated as though redefined as alphanumeric. The INSPECT statement refers to the alphanumeric item.

Each literal must be nonnumeric and can be any figurative constant except ALL.

The comparison operands of the TALLYING phrase (literal-1 or identifier-3, and so on) and/or REPLACING phrase (literal-3 or identifier-5, and so on) are

compared in the left-to-right order specified in the INSPECT statement. A maximum of 15 comparison operands may be specified for each REPLACING and each TALLYING phrase. When the INSPECT statement is contained within IF statements, this maximum number is reduced by the number of nested IF statements.

When the TALLYING/REPLACING operands are the compared operands, the following comparison rules apply:

1. When both the TALLYING and REPLACING phrases are specified, the INSPECT statement is executed as if an INSPECT TALLYING statement were specified and immediately followed by an INSPECT REPLACING statement.
2. The first operand is compared with an equal number of leftmost contiguous characters in the inspected item. The operand matches the inspected characters only if both are equal, character for character.
3. If no match occurs for the first operand, the comparison is repeated for each successive operand until either a match is found or all operands have been acted upon.
4. If a match is found, tallying or replacing takes place as described in TALLYING/REPLACING phrase descriptions. In the inspected item, the first character following the rightmost matching character is now considered the leftmost character position. The process described in comparison rules 2 and 3 is then repeated.
5. If no match is found, the first character in the inspected item following the leftmost inspected character is now considered the leftmost character position. The process described in comparison rules 2 and 3 is then repeated.
6. If the CHARACTERS phrase is specified, an implied 1-character operand is used in the process described in rules 2 and 3. The implied character is considered to always match the inspected character of the item inspected.
7. The actions taken in comparison rules 1 through 6 – which are defined as the comparison cycle – are repeated until the rightmost character in the inspected item has either been matched or has been considered as the leftmost character position. Inspection then terminates.

Figure 5-16 illustrates INSPECT statement comparisons.

INSPECT ID-1 TALLYING ID-2 FOR ALL "*"
REPLACING ALL "***" BY ZEROS.**

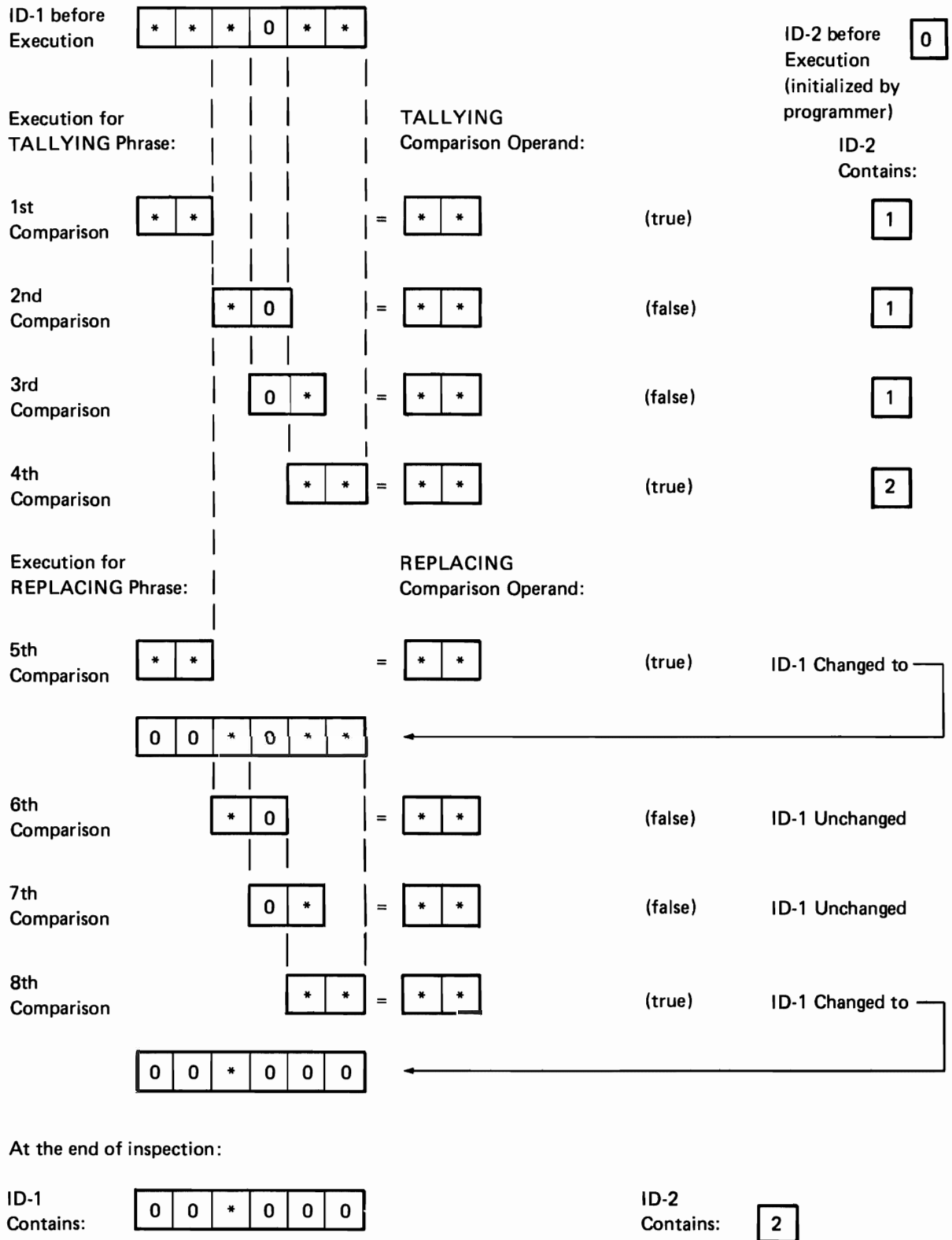


Figure 5-16. INSPECT Statement Execution Results

Note: When the BEFORE/AFTER phrase is specified, the preceding results are modified as described in the BEFORE/AFTER phrase description.

INSPECT

The following example shows an INSPECT statement.


```

1 . . . . . 2 . . . . . 3 . . . . . 4 . . . . . 5 . . . . . 6 . . . . . 7
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ID-1 PIC X(10) VALUE "ACADEMIANS"
01 CONTR-1 PIC 99 VALUE 00.
01 CONTR-2 PIC 99 VALUE ZEROS.
PROCEDURE DIVISION.
* THIS ILLUSTRATES AN INSPECT STATEMENT WITH 2 VARIABLES.
100-BEGIN-PROCESSING.
    DISPLAY CONTR-1 SPACE CONTR-2 UPON MYTUBE.
101-MAINLINE-PROCESSING.
    PERFORM COUNT-IT THRU COUNT-EXIT
    STOP RUN.
COUNT-IT.
    INSPECT ID-1
    TALLYING CONTR-1
        FOR CHARACTERS BEFORE INITIAL "AD"
        CONTR-2
        FOR ALL "MIANS"
DISPLAY-COUNTS.
    DISPLAY "CONTR-1 = " CONTR-1 UPON MYTUBE.
    DISPLAY "CONTR-2 = " CONTR-2 UPON MYTUBE.
    DISPLAY "*****EOJ*****" UPON MYTUBE.
COUNT-EXIT.
EXIT

```

Resultant Output:

```

00 00
CONTR-1 = 02
CONTR-2 = 01
*****EOJ*****

```

TALLYING Phrase

Identifier-2 is the tallying field and must be an elementary integer item defined without the symbol P in its PICTURE character-string. Identifier-2 must be initialized before the INSPECT statement is executed.

Identifier-3 or literal-1 is the comparison operand. If the comparison operand is a figurative constant, it is considered to be a one-character nonnumeric literal.

REPLACING Phrase

Identifier-5 or literal-3 is the comparison operand. Identifier-6 or literal-4 is the replacement field.

The comparison operand and the replacement field must be the same length. The following replacement rules apply:

- If the comparison operand is a figurative constant, it is considered to be a one-character nonnumeric literal. Each character in the inspected item equivalent to the figurative constant is replaced by the single-character replacement field, which must be one character in length.
- If the replacement field is a figurative constant, it is considered to be the same length as the comparison operand. Each nonoverlapping

occurrence of the comparison operand in the inspected item is replaced by the replacement field.

- When the comparison operand and replacement fields are character-strings, each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the character-string specified in the replacement field.
- Once replacement has occurred in a given character position in the inspected item, no further replacement for that character position is made in this execution of the INSPECT statement.

BEFORE/AFTER Phrases

The keywords BEFORE and AFTER should not be used in the same statement.

When either of these phrases is specified, the preceding actions for tallying and replacing are modified.

Identifier-4, identifier-7, literal-2, and literal-5 are delimiters. Tallying and/or replacement of the inspected item is bounded by their presence; however, the delimiters themselves are not counted or replaced.

If the delimiter (literal-2 or literal-5) is a figurative constant, it is considered to be 1 character in length.

In the REPLACING phrase, if the CHARACTERS phrase is specified, the delimiter (literal-5 or identifier-7) must be 1 character in length.

When the BEFORE phrase is specified, tallying and/or replacement of the inspected item begins at the leftmost character and continues until the first occurrence of the delimiter is encountered. If no delimiter is present in the inspected item, tallying and/or replacement continues to the rightmost character.

When the AFTER phrase is specified, tallying and/or replacement of the inspected item begins with the first character to the right of the delimiter and continues to the rightmost character in the inspected item. If no delimiter is present in the inspected item, no tallying or replacement takes place.

When the BEFORE/AFTER phrase is not specified, the following actions take place when the INSPECT TALLYING statement is executed:

- If the ALL phrase is specified, the tallying field is increased by one for each nonoverlapping occurrence in the inspected item of the comparison operand. This process begins at the leftmost character position and continues to the rightmost.
- If the LEADING phrase is specified, the tallying field is increased by one for each contiguous nonoverlapping occurrence of the comparison operand in the inspected item, provided the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which the comparison operand is eligible to participate.

- If the CHARACTERS phrase is specified, the tallying field is increased by one for each character (including the space character) in the inspected item. Thus, execution of the INSPECT TALLYING statement increases the value in the tallying field by the number of characters in the inspected item.

When the BEFORE/AFTER phrase is not specified, the following actions take place when the INSPECT REPLACING statement is executed:

- If the CHARACTERS phrase is specified, the replacement field must be 1 character in length. Each character in the inspected field is replaced by the replacement field. This process begins at the leftmost character and continues to the rightmost.
- If the ALL phrase is specified, each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field, beginning at the leftmost character and continuing to the rightmost.
- If the LEADING phrase is specified, each contiguous nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which this replacement field is eligible to participate.
- If the FIRST phrase is specified, the leftmost occurrence of the comparison operand in the inspected item is replaced by the replacement field.

INSPECT Statement Examples

The following examples illustrate some uses of the INSPECT statement. In all instances, the programmer has initialized the COUNTR field to zero before the INSPECT statement is executed.

INSPECT ID-1 REPLACING CHARACTERS BY ZERO.

ID-1 Before	COUNTR After	ID-1 After
1234567	0	0000000
HIJKLMN	0	0000000

INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS REPLACING CHARACTERS BY SPACES.

ID-1 Before	COUNTR After	ID-1 After
1234567	7	
HIJKLMN	7	

INSPECT ID-1 REPLACING CHARACTERS BY ZEROS BEFORE INITIAL QUOTE.

ID-1 Before	COUNTR After	ID-1 After
456"ABEL	0	000"ABEL
ANDES"12	0	00000"12
"TWAS BR	0	"TWAS BR

INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS AFTER INITIAL "S" REPLACING ALL "A" BY "0".

ID-1 Before	COUNTR After	ID-1 After
ANSELM	3	ONSELM
SACKET	5	SOCKET
PASSED	3	POSSED

INSPECT ID-1 TALLYING COUNTR FOR LEADING "0" REPLACING FIRST "A" BY "2" AFTER INITIAL "C".

ID-1 Before	COUNTR After	ID-1 After
OOACADEMY00	2	OOAC2DEMY00
OOOOALABAMA	4	OOOOALABAMA
CHATAM0000	0	CH2THAM0000

Programming Note: The INSPECT statement is useful for filling all or part of a data item with spaces or zeros. It is also useful for counting the number of times a specific character (for example, zero, space, asterisk) occurs in a data item. In addition, it can be used to translate characters from one collating sequence to another.

MOVE Statement

The MOVE statement transfers data from one area of storage to one or more other areas. The formats of the MOVE statement are as follows:

Format 1

MOVE { identifier-1
literal } TO identifier-2 [, identifier-3] . . .

Format 2

MOVE { CORRESPONDING
CORR } identifier-1 TO identifier-2

General Considerations

Identifier-1 or literal is the sending area. Identifier-2, identifier-3, and so on are the receiving areas.

An index data item cannot be specified in a MOVE statement. Any subscripting or indexing associated with the sending item is evaluated only once: immediately before the data is moved to the first receiving field. Any subscripting or indexing associated with the receiving items is evaluated immediately before the data is moved into the receiving field.

For example, the result of the statement:

```
MOVE A (B) TO B, C (B).
```

is equivalent to

```
MOVE A (B) TO TEMP.  
MOVE TEMP TO B.  
MOVE TEMP TO C (B).
```

where TEMP has been defined as an intermediate result item. The subscript B changed in value between the time the first move took place and the time the final move to C (B) was executed.

After execution of a MOVE statement, a sending field contains the same data as before execution, unless a receiving field overlaps the sending field.

Unexpected results can occur when a redefining item is moved to the redefined item (that is, if B REDEFINES C and the statement MOVE B TO C is executed). Unexpected results can also occur when a redefined item is moved to an item redefining it (from the previous example, unexpected results occur if the statement MOVE C TO B is executed).

Elementary Moves

An elementary move is one in which both the sending and receiving items are elementary items. Each elementary item belongs to one of the following categories:

- **Numeric:** Includes numeric data items, numeric literals, and the figurative constant ZERO/ZEROS/ZEROES
- **Alphabetic:** Includes alphabetic data items and the figurative constant SPACE/SPACES.

Both identifiers following the keyword CORRESPONDING must name group items. In this discussion, these identifiers are referred to as d1 and d2.

A pair of subordinate data items, one from d1 and one from d2, correspond if the following conditions are true:

- At least one of the subordinate items is elementary.

- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.
- The subordinate items are not identified by the keyword FILLER.
- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group item, the items subordinate to it are also ignored. However, d1 and d2 themselves can contain or be subordinate to items containing a REDEFINES or OCCURS clause in their description.

For example, two data hierarchies are defined as follows:

```

05 ITEM-1 OCCURS 6 INDEXED BY X.
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C REDEFINES ITEM-B ...
05 ITEM-2.
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C ...

```

If MOVE CORR ITEM-2 TO ITEM-1(X) is specified, ITEM-A and ITEM-A(X), and ITEM-B and ITEM-B(X) are considered to be corresponding and the moves are performed. ITEM-C and ITEM-C(X) are not included because ITEM-C(X) includes a REDEFINES clause in its data description. ITEM-1 is valid as either d1 or d2.

- Neither d1 nor d2 is described as a level 66, 77 or 88 item, or as a FILLER or USAGE IS INDEX item.

IBM Extension

D1 and/or d2 can be subordinate to a FILLER item.

End of IBM Extension

- Alphanumeric: Includes alphanumeric data items, nonnumeric literals, and all figurative constants except ZERO and SPACE.
- Alphanumeric edited: Includes alphanumeric edited data items.
- Numeric edited: Includes numeric edited data items.

IBM Extension

- Boolean: Includes Boolean data items and Boolean literals.

End of IBM Extension

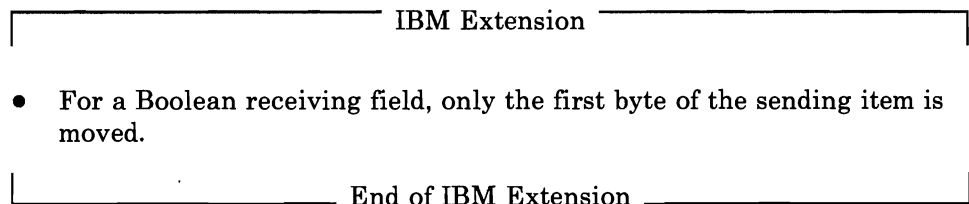
Valid elementary moves are executed according to the following rules:

- Any necessary conversion of data from one form of internal representation to another along with any specified editing in the receiving item takes place during the move.

- For an alphanumeric or alphanumeric edited receiving item:
 - Justification and any necessary space filling take place as described under “Standard Alignment Rules” in Chapter 4. Unused character positions are filled with spaces.
 - If the size of the sending item is greater than the size of the receiving item, excess characters at the right are truncated after the receiving item is filled.
 - If the sending item has an operational sign, the absolute value is used. If the operational sign occupies a separate character, that character is not moved, and the size of the sending item is considered to be one less than its actual size.

- For a numeric or numeric edited receiving item:
 - Alignment by decimal point and any necessary zero filling take place as described under “Standard Alignment Rules” in Chapter 4, except where zeros are replaced because of editing requirements.
 - If the receiving item is signed, the sign of the sending item is placed in the receiving item, with any necessary sign conversion. If the sending item is unsigned, a positive operational sign is generated for the receiving item.
 - The absolute value of the sending item is used if the receiving item has no operational sign.
 - If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, excess digits are truncated.
 - When the sending item is alphanumeric, the data is moved as if the sending item were described as an unsigned integer. It is the user’s responsibility to ensure that the data is numeric.

- For an alphabetic receiving field:
 - Justification and any necessary space filling take place as described under “Standard Alignment Rules” in Chapter 4.
 - If the size of the sending item is greater than the size of the receiving item, excess characters at the right are truncated after the receiving item is filled.



Note: If the receiving field is alphanumeric or numeric edited, and the sending field is a scaled integer (that is, it has a P as the rightmost character in its PICTURE character-string), the scaling positions are treated as trailing zeros when the MOVE statement is executed.

Figure 5-17 shows valid and invalid elementary moves for each category.

Group Moves

A group move is one in which one or both of the sending and receiving fields are a group item. A group move is treated exactly as though it were an alphanumeric elementary move except that data is not converted from one form of internal representation to another. In a group move, the receiving area is filled without consideration for the individual elementary items contained within either the sending area or the receiving area. See "OCCURS Clause" in Chapter 6 for additional information.

Sending Item Category	Receiving Item Category						
	Alphabetic	Alphanumeric	Alphanumeric Edited	Numeric Integer	Numeric Noninteger	Numeric Edited	Boolean
Alphabetic and SPACE	YES	YES	YES	NO	NO	NO	NO
Alphanumeric	YES	YES	YES	YES ⁴	YES ⁴	YES ⁴	YES ⁵
Nonnumeric Literal	YES	YES	YES	YES ¹	YES ¹	YES ¹	YES ⁵
Alphanumeric Edited	YES	YES	YES	NO	NO	NO	NO
Numeric Integer ²	NO	YES	YES	YES	YES	YES	NO
Numeric Noninteger ²	NO	NO	NO	YES	YES	YES	NO
Numeric Edited	NO	YES	YES	NO	NO	NO	NO
LOW/HIGH-VALUES QUOTES	NO	YES	YES	NO	NO	NO	NO
ZERO	NO	YES	YES	YES	YES	YES	YES
Boolean ³	NO	YES	YES	NO	NO	NO	YES

YES = move is valid
 NO = move is invalid

¹ Moved only if an unsigned integer.
² Includes numeric literals.
³ Includes Boolean literals.
⁴ Compiler assumes alphanumeric item is an unsigned integer.
⁵ Compiler assumes item is a 0 or a 1.

Figure 5-17. Valid and Invalid Elementary Moves

Format 1 Considerations

When Format 1 is specified, the identifiers can be either group or elementary items. The data in the sending area is moved into the first receiving area (identifier-2); then it is moved from the sending area into the second receiving area (identifier-3), and so on.

Format 2 Considerations

CORRESPONDING Phrase: The CORRESPONDING phrase allows data to be moved between elementary items of the same name simply by specifying the group items to which they belong.

The abbreviation CORR is equivalent to the keyword CORRESPONDING.

SET Statement

IBM Extension

The SET statement is used to alter the status of external switches and the values of conditional variables. See "SET Statement" in Chapter 6 for the other two formats.

Format 1

```
SET mnemonic-name-1 [ , mnemonic-name-2 ] . . . TO { ON  
OFF }
```

Format 2

```
SET condition-name-1 [ , condition-name-2 ] . . . TO TRUE
```

For Format 1 each mnemonic-name must be associated with an external switch, the status of which can be altered. The only external switches allowed are the UPSI switches, UPSI-0 through UPSI-7.

The status of each external switch associated with the specified mnemonic-name is modified such that the truth value resultant from evaluation of a condition-name associated with that switch will reflect an on status if the ON phrase is specified, or an off status if the OFF phrase is specified. For additional information, refer to "Simple Conditions, Switch-Status Condition" earlier in this chapter.

Format 2 allows conditional items to be set to their stated values. The literal in the VALUE clause associated with the condition-name is moved to the conditional variable according to the rules for elementary moves.

If multiple condition-names are specified, the results are the same as those obtained if a separate SET statement were written for each condition-name in the same order specified in the SET statement.

End of IBM Extension

STRING Statement

The STRING statement gives the programmer the ability to concatenate the partial or complete contents of two or more data items into a single data item.

Format

$$\begin{aligned} & \underline{\text{STRING}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left[\begin{array}{l} \text{, identifier-2} \\ \text{, literal-2} \end{array} \right] \dots \underline{\text{DELIMITED BY}} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{literal-3} \\ \underline{\text{SIZE}} \end{array} \right\} \\ & \left[\begin{array}{l} \text{, } \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-4} \end{array} \right\} \left[\begin{array}{l} \text{, identifier-5} \\ \text{, literal-5} \end{array} \right] \dots \underline{\text{DELIMITED BY}} \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-6} \\ \underline{\text{SIZE}} \end{array} \right\} \end{array} \right] \dots \\ & \underline{\text{INTO}} \text{ identifier-7} \left[\underline{\text{WITH POINTER}} \text{ identifier-8} \right] \\ & \left[\underline{\text{ON OVERFLOW}} \text{ imperative-statement} \right] \end{aligned}$$

Each literal must be a nonnumeric literal; each can be any figurative constant except the ALL literal. When a figurative constant is specified, it is considered a 1-character nonnumeric literal.

All identifiers except identifier-8 (the pointer item) must have USAGE DISPLAY, explicitly or implicitly.

The sending fields are identifier-1, identifier-2, identifier-4, identifier-5, or their corresponding literals.

The receiving field is identifier-7, which must be an elementary alphanumeric item without editing symbols and without the JUSTIFIED clause in its description.

The delimiters are identifier-3, identifier-6, or their corresponding literals, or the keyword SIZE. The delimiters specify the character(s) delimiting the data to be transferred; when SIZE is specified, the complete sending area is transferred.

When the sending field or any of the delimiters are elementary numeric items, they must be described as integers, and their PICTURE character-strings must not contain the symbol P.

The pointer field is identifier-8, which must be an elementary integer data item large enough to contain a value equal to the length of the receiving area plus one. The pointer field must not contain the symbol P in its PICTURE character-string.

STRING Statement Execution

When the STRING statement is executed, data is transferred from the sending fields to the receiving field. The order in which sending fields are processed is the order in which they are specified. The following rules apply:

- Characters from the sending fields are transferred to the receiving field according to the rules for alphanumeric to alphanumeric elementary moves except that no space filling is provided.
- When the DELIMITED BY identifier/literal is specified, the contents of each sending item are transferred character by character beginning with the leftmost and continuing until either a delimiter for this sending field is reached (the delimiter itself is not transferred) or the rightmost character of this sending field has been transferred.
- When DELIMITED BY SIZE is specified, each sending field is transferred in its entirety to the receiving field.
- When the receiving field is filled or when all the sending fields have been processed, the operation is ended.
- When the POINTER phrase is specified, an explicit pointer field is available to the COBOL user to control placement of data in the receiving field. The user must set the explicit pointer's initial value, which must not be less than one and not more than the character count of the receiving field. The pointer field must be defined as large enough to contain a value equal to the length of the receiving field plus 1; this precludes arithmetic overflow when the system updates the pointer at the end of the transfer.
- When the POINTER phrase is not specified, no pointer is available to the user. However, an implicit pointer with an initial value of one is used by the system.
- When the STRING statement is executed, the initial pointer value (explicit or implicit) points to the first character position within the receiving field into which data is to be transferred. Beginning at that position, data is then positioned character by character from left to right. After each character is positioned, the explicit or implicit pointer is incremented by one. The value in the pointer field is changed only in this manner. At the end of processing, the pointer value always indicates one character beyond the last character transferred into the receiving field.
- If, at any time during or after initiation of STRING statement execution, the pointer value (explicit or implicit) is less than one or exceeds a value equal to the length of the receiving field, no more data is transferred into the receiving field and, if specified, the ON OVERFLOW imperative-statement is executed. (The ON OVERFLOW statement is not executed unless there was an attempt to move in one or more characters beyond the end of identifier-7.)

- If the ON OVERFLOW phrase is not specified, then when the preceding conditions occur, control passes to the next executable statement.

After STRING statement execution is completed, only that part of the receiving field into which data was transferred is changed. The rest of the receiving field contains the data that was present before this execution of the STRING statement. Figure 5-18 illustrates the rules of execution for the STRING statement.

STRING Statement Example

The following example illustrates some of the considerations that apply to the STRING statement.

In the Data Division, the programmer has defined the following fields:

```

01 RPT-LINE      PICTURE X(120).
01 LINE-POS     PICTURE 99.
01 LINE-NO      PICTURE 9(5) VALUE 1.
01 DEC-POINT    PICTURE X VALUE ".".

```

In the File Section, he has defined the following input record:

```

01 RCD-01.
05 CUST-INFO.
10 CUST-NAME    PICTURE X(15).
10 CUST-ADDR    PICTURE X(34).
05 BILL-INFO.
10 INV-NO       PICTURE X(6).
10 INV-AMT      PICTURE $$, $$$ .99.
10 AMT-PAID     PICTURE $$, $$$ .99.
10 DATE-PAID    PICTURE X(8).
10 BAL-DUE      PICTURE $$, $$$ .99.
10 DATE-DUE     PICTURE X(8).

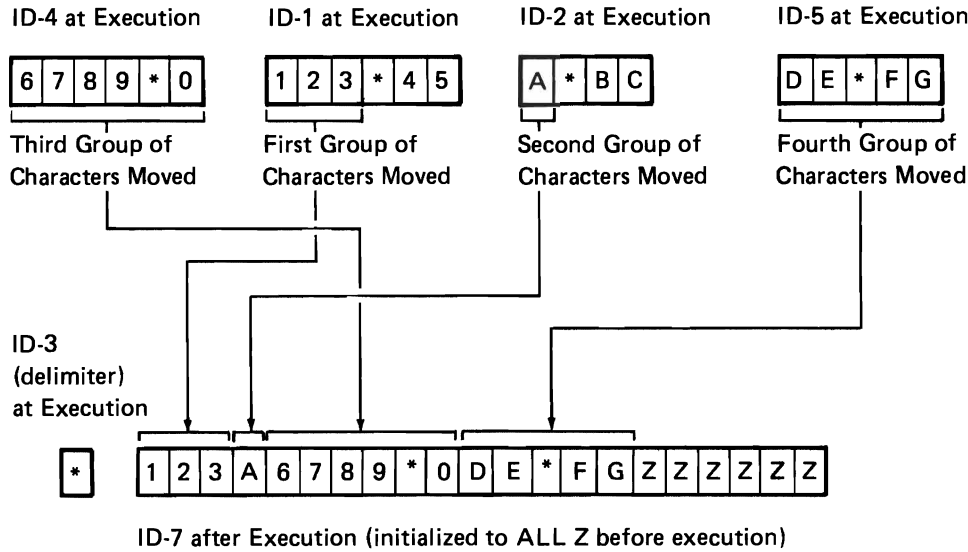
```

STRING Statement to be Executed:

```

STRING ID-1 ID-2 DELIMITED BY ID-3
      ID-4 ID-5 DELIMITED BY SIZE
      INTO ID-7 WITH POINTER ID-8.
  
```

Results:



ID-8
(pointer)
after Execution

1 6

(initialized to 01 before execution)

Figure 5-18. STRING Statement Execution Results

The user wants to construct an output line consisting of portions of the information from RCD-01. The line is to consist of a line number, customer name and address, invoice number, date due, and balance due, truncated to the dollar figure shown.

The record as read in contains the following information:

```

J.B. bSMITH bbbbb
444 bSPRING bST., bCHICAGO, bILL. bbbbb
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
10/22/76
  
```

In the Procedure Division, the user initializes RPT-LINE to SPACES and sets LINE-POS (which is to be used as the pointer field) to 4. Then he issues this STRING statement:

```

STRING LINE-NO SPACE
      CUST-INFO SPACE
      INV-NO SPACE
      DATE-DUE SPACE
      DELIMITED BY SIZE,
      BAL-DUE
      DELIMITED BY DEC-POINT
      INTO RPT-LINE
      WITH POINTER LINE-POS.

```

When the statement is executed, the following actions take place:

1. The field LINE-NO is moved into positions 4 through 8 of RPT-LINE.
2. A space is moved into position 9.
3. The group item CUST-INFO is moved into positions 10 through 58.
4. A space is moved into position 59.
5. INV-NO is moved into positions 60 through 65.
6. A space is moved into position 66.
7. DATE-DUE is moved into positions 67 through 74.
8. A space is moved into position 75.
9. The portion of BAL-DUE that precedes the decimal point is moved into positions 76 through 81.

After the STRING statement has been executed:

- RPT-LINE appears as shown in Figure 5-19.
- LINE-POS contains the value 82.

Programming Note: One STRING statement can be written instead of a series of MOVE statements.

Column			
4	10	25	
↓	↓	↓	
00001	J.B. SMITH	444	SPRING ST., CHICAGO, ILL.
60	67	76	
↓	↓	↓	
A14725	10/22/76	\$2,336	

Figure 5-19. STRING Statement Example Output Data

UNSTRING Statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

Format

UNSTRING identifier-1

```
[ DELIMITED BY [ ALL ] { identifier-2 } [ , OR [ ALL ] { identifier-3 } ] . . . ]
INTO identifier-4 [ , DELIMITER IN identifier-5 ] [ , COUNT IN identifier-6 ]
[ , identifier-7 [ , DELIMITER IN identifier-8 ] [ , COUNT IN identifier-9 ] ] . . .
[ WITH POINTER identifier-10 ] [ TALLYING IN identifier-11 ]
[ ON OVERFLOW imperative-statement ]
```

Each literal must be a nonnumeric literal; each may be any figurative constant except ALL literal. When a figurative constant is specified, it is considered to be a 1-character nonnumeric literal.

Sending Field

Identifier-1 is the sending field. It must be an alphanumeric data item. Data is transferred from this field to the receiving fields.

DELIMITED BY Phrase: This phrase specifies delimiters within identifier-1 that control the data transfer.

The delimiters are identifier-2, identifier-3, or their corresponding literals. Each identifier or literal specified represents one delimiter. No more than 30 delimiters can be specified. Each must be an alphanumeric data item.

If a delimiter contains two or more characters, it is recognized in the sending field only if the delimiter characters are contiguous and, in the sequence specified, in the delimiter item.

When two or more delimiters are specified, an OR condition exists and each nonoverlapping occurrence of any one of the delimiters is recognized in the sending field in the sequence specified. For example, if DELIMITED BY AB OR BC is specified, then an occurrence of either AB or BC in the sending field is considered a delimiter. An occurrence of ABC is considered an occurrence of AB, and the search for another delimiter resumes with C.

When the DELIMITED BY ALL phrase is not specified, and two or more contiguous occurrences of any delimiter are encountered, the current data receiving field is filled with spaces or zeros according to the description of the data receiving field.

When the DELIMITED BY ALL phrase is specified, one or more contiguous occurrences of any delimiter are treated as if they were only one occurrence, and this one occurrence is moved to the delimiter receiving field (if specified). The delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved into the current delimiter receiving field according to the rules of the MOVE statement.

The DELIMITER IN and COUNT IN phrases can be specified only if the DELIMITER BY phrase is specified.

Data Receiving Fields

Identifier-4, identifier-7, and so on, are the data receiving fields and must have USAGE DISPLAY. These fields can be defined as:

- Alphabetic (without the symbol B in the PICTURE string)
- Alphanumeric
- Numeric (without the symbol P in the PICTURE string).

These fields must not be defined as alphanumeric edited or numeric edited items. Data is transferred to these fields from the sending field.

DELIMITER IN Phrase: The delimiter receiving fields are identifier-5, identifier-8, and so on. These identifiers must be alphanumeric. They must not be defined as alphanumeric edited or numeric edited items.

COUNT IN Phrase: The data-count fields for each data transfer are identifier-6, identifier-9, and so on. These identifiers must be described as elementary numeric integer data items; they cannot contain the symbol P in their PICTURE clauses. Each field holds the count of delimited characters in the sending field to be transferred to this receiving field; the delimiters are not included in this count.

POINTER Phrase: The pointer field is identifier-10. This identifier must be described as an elementary numeric integer data item; it cannot contain the symbol P in its PICTURE clause. The identifier contains a value position in the sending field. When this phrase is specified, this field must be initialized before execution of the UNSTRING statement to a value that is not less than one and not greater than the count of the sending field.

TALLYING Phrase: The field-count is identifier-11. This identifier must be described as an elementary numeric integer data item; it cannot contain the symbol P in its PICTURE clause. The identifier is incremented by the number of data receiving fields acted upon in this execution of the UNSTRING statement. When this phrase is specified, this field must be initialized before execution of the UNSTRING statement.

The data-count fields, the pointer field, and the field-count field must each be integer items without the symbol P in the PICTURE character-strings.

UNSTRING Statement Execution

When the UNSTRING statement is initiated, the current data receiving field is identifier-4. Data is transferred from the sending field to the current data receiving field according to the following rules:

- If the POINTER phrase is not specified, the sending field character-string is examined beginning with the leftmost character. If the POINTER phrase is specified, the field is examined beginning at the relative character position specified by the value in the pointer field.
- If the DELIMITED BY phrase is specified, the examination proceeds left to right character by character until a delimiter is encountered. If the end of the sending field is reached before a delimiter is found, the examination ends with the last character in the sending field.
- If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current data receiving field, which depends on its data category:
 - If the receiving field is alphanumeric or alphabetic, the number of characters examined is equal to the number of characters in the current receiving field.
 - If the receiving field is numeric, the number of characters examined is equal to the number of characters in the integer portion of the current receiving field.
 - If the receiving field is described with the SIGN IS SEPARATE clause, the characters examined are one fewer than the size of the current receiving field.
 - If the receiving field is described as a variable-length data item, the number of characters examined is determined by the current size of the current receiving field.
- The examined characters (excluding any delimiter characters) are treated as an alphanumeric elementary item, and are moved into the current data receiving field according to the rules for the MOVE statement.
- If the DELIMITER IN phrase is specified, the delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved to the current delimiter receiving field according to the rules for the MOVE statement. If the delimiting condition is the end of the sending field, the current delimiter receiving field is filled with spaces.
- If the COUNT IN phrase is specified, a value equal to the number of examined characters (excluding any delimiters) is moved into the data count field, according to the rules for an elementary move.
- If the DELIMITED BY phrase is specified, the sending field is further examined, beginning with the first character to the right of the delimiter.

- If the DELIMITED BY phrase is not specified, the sending field is further examined, beginning with the first character to the right of the last character examined.
- After data is transferred to the first data receiving field (identifier-4), the current data receiving field becomes identifier-7. For each succeeding current data receiving field, the preceding procedure is repeated – either until all of the characters in the sending field have been transferred, or until there are no more unfilled data receiving fields.
- When the POINTER phrase is specified, the contents of the pointer field behaves as if incremented by one for each examined character in the sending field. When this execution of the UNSTRING statement is completed, the pointer field contains a value equal to its initial value plus the number of characters examined in the sending field.
- When the TALLYING phrase is specified and the execution of the UNSTRING statement is completed, the tallying identifier contains a value equal to the initial value plus the number of data receiving areas acted upon; this count includes any null fields.
- When an overflow condition exists, the execution of the UNSTRING statement is terminated. If the ON OVERFLOW phrase has been specified, that imperative-statement is executed. If the ON OVERFLOW phrase has not been specified, control passes to the next executable statement. An overflow condition exists when:
 - An UNSTRING statement is initiated and the value in the pointer field is less than 1 or greater than the length of the sending field.
 - Or, all data receiving fields have been acted upon during UNSTRING statement execution, and the sending field still contains unexamined characters.

Note: If any of the UNSTRING statement identifiers are subscripted or indexed, the subscripts and indexes are evaluated as follows:

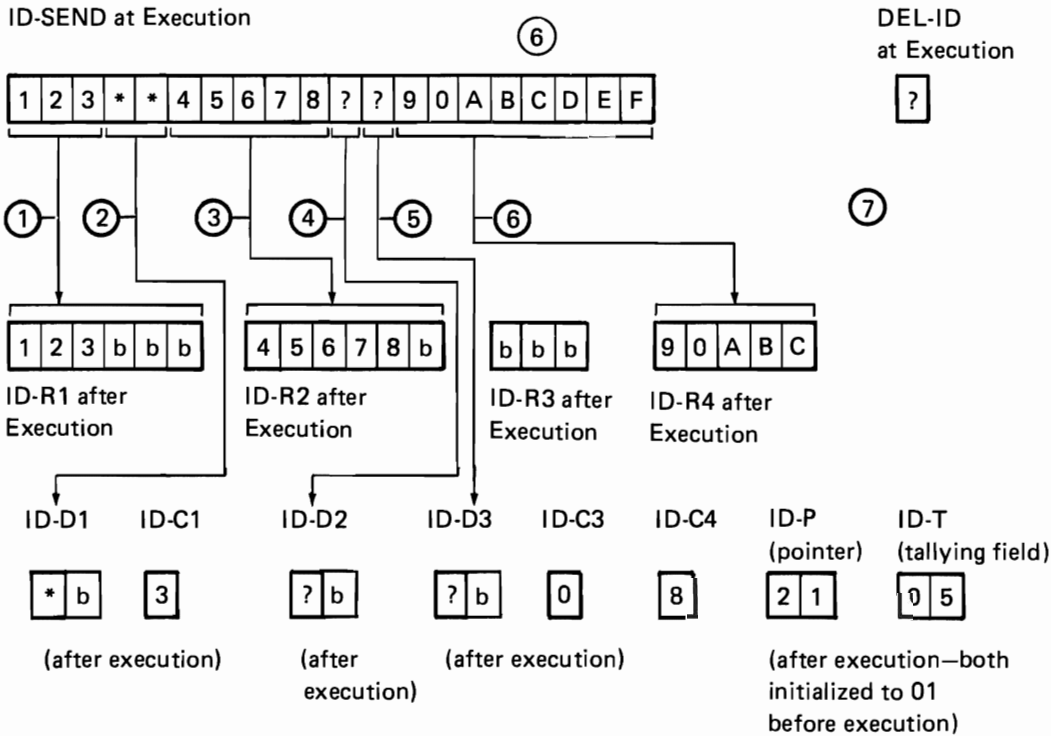
- Any subscripting or indexing associated with the sending field, the pointer field, or the field-count field is evaluated only once – immediately before any data is transferred.
- Any subscripting or indexing associated with the delimiters, the data and delimiter receiving fields or the data-count fields, is evaluated immediately before the transfer of data into the affected data item.

Figure 5-20 illustrates the rules of execution for the UNSTRING statement.

The following UNSTRING statement has the execution results shown:

```
UNSTRING ID-SEND DELIMITED BY DEL-ID OR ALL "*"
      INTO ID-R1 DELIMITER IN ID-D1 COUNT IN ID-C1
           ID-R2 DELIMITER IN ID-D2
           ID-R3 DELIMITER IN ID-D3 COUNT IN ID-C3
           ID-R4 COUNT IN ID-C4
      WITH POINTER ID-P
      TALLYING IN ID-T
      ON OVERFLOW GO TO OFLOW-EXIT.
```

(All the data receiving fields are defined as alphanumeric.)



The order of execution is:

- Three characters are examined before a delimiter is encountered. These characters are moved to ID-R1.
- The delimiter "*" is placed in ID-D1; the number of characters before the delimiter is moved to ID-C1.

Note: Because ALL "*" is specified, the second asterisk is treated as part of the delimiter. Only the first occurrence of the delimiter is moved to the DELIMITER IN data item, ID-D1.

- Five characters are examined before the next delimiter is encountered. These characters are moved to ID-R2.

- The delimiter "?" is placed in ID-D2.

Note: Because ALL is not specified for this delimiter, only the first "?" is

considered to be the delimiter. The second "?" is the delimiter for the next receiving field.

- No characters are examined before the next delimiter is encountered, so no characters are moved to ID-R3. Padding causes ID-R3 to be filled with spaces. The delimiter "?" is placed in ID-D3. The number of examined characters (0) is moved to ID-C3.

- The remaining 8 characters are examined; no delimiter is found. These characters are moved to ID-R4. Because of the size of the receiving field, the last 3 characters are lost due to truncation. However, the total number of characters examined (8) is moved to ID-C4.

- After execution, ID-P is incremented by the number of characters examined, and ID-T is incremented by the number of receiving fields processed.

Figure 5-20. UNSTRING Statement Execution Results

UNSTRING Statement Example

The following example illustrates some of the considerations that apply to the UNSTRING statement.

In the Data Division, the user has defined the following input record to be acted upon by the UNSTRING statement:

```
01 INV-RCD.  
05 CONTROL-CHARS PIC XX.  
05 ITEM-INDENT PIC X(20).  
05 FILLER PIC X.  
05 INV-CODE PIC X(10)  
05 FILLER PIC X.  
05 NO-UNITS PIC 9(6).  
05 FILLER PIC X.  
05 PRICE-PER-M PIC 99999.  
05 FILLER PIC X.  
05 RTL-AMT PIC 9(6).99.
```

The next two records are defined as receiving fields for the UNSTRING statement. DISPLAY-REC is to be used for printed output. WORK-REC is to be used for further internal processing.

```
01 DISPLAY-REC  
05 INV-NO PIC X(6).  
05 FILLER PIC X VALUE SPACE  
05 ITEM-NAME PIC X(20).  
05 FILLER PIC X VALUE SPACE  
05 DISPLAY-DOLS PIC 9(6).  
  
01 WORK-REC  
05 M-UNITS PIC 9(6).  
05 FIELD-A PIC 9(6).  
05 WK-PRICE  
REDEFINES  
FIELD-A PIC 9999V99  
05 INV-CLASS PIC X(3).
```

The user has also defined the following fields for use as control fields in the UNSTRING statement.

```
01 DBY-1 PIC X, VALUE IS "."  
01 CTR-1 PIC 99, VALUE IS ZERO.  
01 CTR-2 PIC 99, VALUE IS ZERO  
01 CTR-3 PIC 99, VALUE IS ZERO.  
01 CTR-4 PIC 99, VALUE IS ZERO.  
01 DLTR-1 PIC X  
01 DLTR-2 PIC X.  
01 CHAR-CT PIC 99, VALUE IS 3.  
01 FLDS-FILLED PIC 99, VALUE IS ZERO.
```

In the Procedure Division, the user writes the following UNSTRING statement to move subfields of INV-RCD to the subfields of DISPLAY-REC and WORK-REC:

```

UNSTRING INV-RCD
  DELIMITED BY ALL SPACES
    OR "/"
    OR DBY-1
  INTO ITEM-NAME COUNT IN CTR-1,
  INV-NO DELIMITER IN DLTR-1
    COUNT IN CTR-2,
  INV-CLASS,
  M-UNITS COUNT IN CTR-3,
  FIELD-A,
  DISPLAY-DOLS DELIMITER IN DLTR-2
    COUNT IN CTR-4
  WITH POINTER CHAR-CT
  TALLYING IN FLDS-FILLED
  ON OVERFLOW
  GO TO UNSTRING-COMPLETE.

```

Before the UNSTRING statement is issued, the user places the value 3 in the CHAR-CT (the pointer item), so as not to work with the two control characters at the beginning of INV-RCD. In DBY-1, a period is placed for use as a delimiter, and in FLDS-FILLED (the tallying item) the value 0 is placed. The following data is then read into INV-RCD as shown in Figure 5-21.

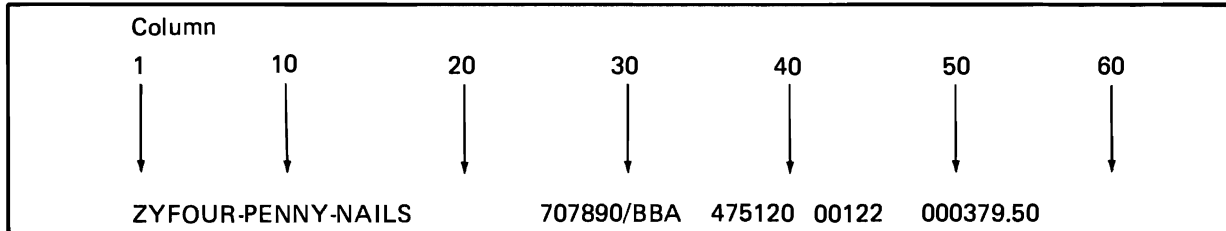


Figure 5-21. UNSTRING Statement Example – Input Data

When the UNSTRING statement is executed, the following actions take place:

1. Positions 3 through 18 (FOUR-PENNY-NAILS) of INV-RCD are placed in ITEM-NAME, left-justified within the area, and the unused character positions are padded with spaces. The value 16 is placed in CTR-1.
2. Because ALL SPACES is specified as a delimiter, the five contiguous SPACE characters are considered to be one occurrence of the delimiter.
3. Positions 24 through 29 (707890) are placed in INV-NO. The delimiter character / is placed in DLTR-1, and the value 6 is placed in CTR-2.
4. Positions 31 through 33 are placed in INV-CLASS. The delimiter is a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is merely bypassed.
5. Positions 35 through 40 (475120) are examined and are placed in M-UNITS. The delimiter is a SPACE, but because no receiving field has been defined as a receiving area for delimiters, the SPACE is bypassed. The value 6 is placed in CTR-3.
6. Positions 42 through 46 (00122) are placed in FIELD-A and right-justified within the area. The high-order digit position is filled with a 0 (zero). The delimiter is a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is bypassed.

7. Positions 48 through 53 (000379) are placed in DISPLAY-DOLS. The period delimiter character is placed in DLTR-2, and the value 6 is placed in CTR-4.
8. Because all receiving fields have been acted upon and two characters of data in INV-RCD have not been examined, the ON OVERFLOW exit is taken, and execution of the UNSTRING statement is completed.

At the end of execution of the UNSTRING statement, DISPLAY-REC contains the following data:

```
707890 FOUR-PENNY-NAILS 000379
```

WORK-REC contains the following data:

```
475120000122BBA
```

CHAR-CT (the pointer field) contains the value 55, and FLD-FILLED (the tallying field) contains the value 6.

Programming Note: One UNSTRING statement can be written instead of a series of MOVE statements.

Procedure Branching Statements

Statements, sentences, and paragraphs in the Procedure Division are ordinarily executed sequentially. The procedure branching statements allow alterations in the sequence. The procedure branching statements are: ALTER, EXIT, GO TO, PERFORM, and STOP.

ALTER Statement

The ALTER statement changes the transfer point specified in a GO TO statement.

Format

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2  
[ , procedure-name-3 TO [PROCEED TO] procedure-name-4 ] . . .
```

Procedure-name-1, procedure-name-3, and so on, must each name a Procedure Division paragraph that contains only one sentence. That sentence must be a GO TO statement without the DEPENDING ON phrase.

Procedure-name-2, procedure-name-4, and so on, must each name a Procedure Division section or paragraph.

ALTER statement execution modifies the GO TO statement in the paragraph named by procedure-name-1, procedure-name-3, and so on. Subsequent executions of the modified GO TO statement(s) cause control to be transferred to procedure-name-2, and (if specified) procedure-name-4, and so on. For example:

```
PARAGRAPH-1.  
  GO TO BYPASS-PARAGRAPH.  
PARAGRAPH-1A.  
.  
BYPASS-PARAGRAPH.  
.  
  ALTER PARAGRAPH-1 TO PROCEED TO  
    PARAGRAPH-2.  
.  
PARAGRAPH-2.  
.  
.
```

Before the ALTER statement is executed, when control reaches PARAGRAPH-1, the GO TO statement transfers control to BYPASS-PARAGRAPH. After execution of the ALTER statement, however, the next time control reaches PARAGRAPH-1, the GO TO statement transfers control to PARAGRAPH-2.

Programming Note: The ALTER statement acts as a program switch, allowing, for example, one sequence of execution during initialization and another sequence during the bulk of file processing. Because altered GO TO statements are difficult to debug, it is preferable to test a switch, and based on the value of the switch, execute a particular code sequence.

Segmentation Information

A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number. All other uses of the ALTER statement are valid and are performed.

Modified GO TO statements in independent segments can sometimes be returned to their initial states. See "Program Segments, Independent Segments" in Chapter 6 for further discussion.

EXIT Statement

The EXIT statement provides a common end point for a series of procedures.

Format

EXIT [PROGRAM] .

The EXIT statement must appear in a sentence by itself, and this sentence must be the only sentence in the paragraph. The EXIT statement lets the user assign a procedure-name to a given point in a program.

The EXIT statement has no other effect on the compilation or execution of the program.

The EXIT PROGRAM statement is discussed under “Procedure Division – Inter-Program Communication” in Chapter 6.

Programming Note: The EXIT statement is useful for documenting the end point in a series of procedures. If an exit paragraph is written as the last paragraph in a Declarative procedure or a series of performed procedures, it identifies the point at which control will be transferred. When control reaches such an exit paragraph and the associated Declarative or PERFORM statement is active, control is transferred to the appropriate part of the Procedure Division. When control reaches such an exit paragraph and no associated PERFORM statement or Declarative procedure is active, control passes through the EXIT statement to the first statement of the next paragraph.

If an EXIT statement is not written, the end of the sequence is difficult to determine unless the user knows the logic of the program.

GO TO Statement

The GO TO statement transfers control from one part of the Procedure Division to another. The formats of the GO TO statement are as follows:

Format 1

GO TO [procedure-name-1]

Format 2

GO TO procedure-name-1 [, procedure-name-2] . . . , procedure-name-n
DEPENDING ON identifier

Each procedure-name specified must name a paragraph or section in the Procedure Division.

Format 1 – Unconditional GO TO

The GO TO statement transfers control to the first statement in the paragraph or section named in procedure-name-1 unless the GO TO statement has been modified by an ALTER statement.

When a Format 1 GO TO statement appears in a sequence of imperative statements, it must be the last statement in the sequence.

When a paragraph is referred to by an ALTER statement, the paragraph can consist only of a paragraph-name followed by a Format 1 GO TO statement.

If procedure-name-1 is not specified in a Format 1 GO TO statement, an ALTER statement must have been executed before the execution of the GO TO statement. The GO TO statement must immediately follow a paragraph-name and must be the only statement in the paragraph.

Format 2 – Conditional GO TO

Control is transferred to one of a series of procedures, depending on the value of identifier. Identifier must name an elementary integer item. When identifier has a value of one, control is transferred to the first statement in the procedure named by procedure-name-1; if it has a value of two, control is transferred to the first statement in the procedure named by procedure-name-2, and so on.

If the value of identifier is anything other than a value within the range 1 through n (where n is the number of procedure-names specified in this GO TO statement), the GO TO statement is ignored. Instead, control passes to the next statement in the normal sequence of execution.

The maximum number of procedure-names permitted for a Format 2 GO TO statement is 255.

PERFORM Statement

The PERFORM statement transfers control explicitly to one or more procedures and implicitly returns control to the next executable statement after execution of the specified procedure(s) is completed. The formats of the PERFORM statement are as follows:

Format 1

$$\underline{\text{PERFORM}} \text{ procedure-name-1 } \left[\left\{ \begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right\} \text{ procedure-name-2} \right]$$

Format 2

$$\underline{\text{PERFORM}} \text{ procedure-name-1 } \left[\left\{ \begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right\} \text{ procedure-name-2} \right] \left\{ \begin{array}{c} \text{identifier-1} \\ \text{integer-1} \end{array} \right\} \underline{\text{TIMES}}$$

Format 3

PERFORM procedure-name-1 [{ THROUGH } procedure-name-2] UNTIL condition-1

Format 4

PERFORM procedure-name-1 [{ THROUGH } procedure-name-2]
VARYING { identifier-1 } FROM { identifier-2 }
 { index-name-1 } { index-name-2 }
 { literal-2 }
BY { identifier-3 } UNTIL condition-1
 { literal-3 }
[AFTER { identifier-4 } FROM { identifier-5 }
 { index-name-4 } { index-name-5 }
 { literal-5 }
BY { identifier-6 } UNTIL condition-2
 { literal-6 }
[AFTER { identifier-7 } FROM { identifier-8 }
 { index-name-7 } { index-name-8 }
 { literal-8 }
BY { identifier-9 } UNTIL condition-3]]

Each procedure-name must name a section or paragraph in the Procedure Division.

When both procedure-name-1 and procedure-name-2 are specified, if either is a procedure-name in a Declarative procedure, then both must be procedure-names in the same Declarative procedure.

Each identifier must name a numeric elementary item.

Each literal must be a numeric literal.

The set of statements within the range of procedure-name-1 (through procedure-name-2 if specified) for a PERFORM statement are referred to as the *specified set of statements*.

Whenever a PERFORM statement is executed, control is transferred to the first statement of the specified set of statements. Control is always returned to the statement following the PERFORM statement. The point from which this control is returned is determined as follows:

- If procedure-name-1 is a paragraph name and procedure-name-2 is not specified, the return is made after the execution of the last statement of procedure-name-1.
- If procedure-name-1 is a section-name and procedure-name-2 is not specified, the return is made after the execution of the last sentence of the last paragraph in that section.
- If procedure-name-2 is specified and it is a paragraph name, the return is made after the execution of the last statement of that paragraph.
- If procedure-name-2 is specified and it is a section-name, the return is made after the execution of the last sentence of the last paragraph in the section.

The only necessary relationship between procedure-name-1 and procedure-name-2 is that a consecutive sequence of operations is executed beginning at the procedure named by procedure-name-1 and ending with the execution of the procedure named by procedure-name-2.

When both procedure-name-1 and procedure-name-2 are specified, GO TO and PERFORM statements can appear within the sequence of statements contained in these paragraphs or sections. A GO TO statement should not refer to a procedure-name outside the range of procedure-name-1 through procedure-name-2. If this is done, results are unpredictable and are not diagnosed.

When only procedure-name-1 is specified, PERFORM and GO TO statements can appear within the procedure. A GO TO statement should not refer to a procedure-name outside the range of procedure-name-1. If this is done, results are unpredictable and are not diagnosed.

When the performed procedures include another PERFORM statement, the sequence of procedures associated with the embedded PERFORM statement must be totally included in or totally excluded from the performed procedures of the first PERFORM statement. That is, an active PERFORM statement whose execution point begins within the range of performed procedures of another active PERFORM statement must not allow control to pass through the exit point of the other active PERFORM statement. In addition, two or more such active PERFORM statements must not have a common exit.

IBM Extension

Two active PERFORM statements can have a common exit point.

End of IBM Extension

When control passes to the sequence of procedures by means other than a PERFORM statement, control passes through the exit point to the next executable statement as if no PERFORM statement referred to these procedures.

The range of a PERFORM statement logically consists of all executed statements resulting from the execution of a PERFORM statement, and includes the implicit transfer of control to the end of the PERFORM statement. This range includes all executed statements in Declarative procedures as well as statements resulting from the transfer of control by CALL, EXIT without the PROGRAM phrase, GO TO, and PERFORM statements. The statements in the range of a PERFORM statement need not appear consecutively in the source program.

Figure 5-22 illustrates valid sequences of execution for PERFORM statements.

The preceding rules refer to all four formats of the PERFORM statement. The following sections give rules applying to each individual format.

Format 1

Format 1 is the basic PERFORM statement. The procedure(s) referred to is executed once, and then control passes to the next executable statement following the PERFORM statement.

Format 2

Format 2 uses the TIMES phrase. Identifier-1 must name an integer item. The procedure(s) referred to is executed the number of times specified by the value in identifier-1 or integer-1. Control then passes to the next executable statement following the PERFORM statement. The following rules apply:

- If identifier-1 is zero or a negative number at the time the PERFORM statement is initiated, control passes to the statement following the PERFORM statement.
- After the PERFORM statement has been initiated, any reference to identifier-1 or change in the value of identifier-1 has no effect in varying the number of times the procedures are executed.

Format 3

Format 3 uses the UNTIL phrase. The procedure(s) referred to is performed until the condition specified by the UNTIL phrase is true. Control is then passed to the next executable statement following the PERFORM statement.

If condition-1 is true at the time the PERFORM statement is encountered, the specified procedure(s) is not executed.

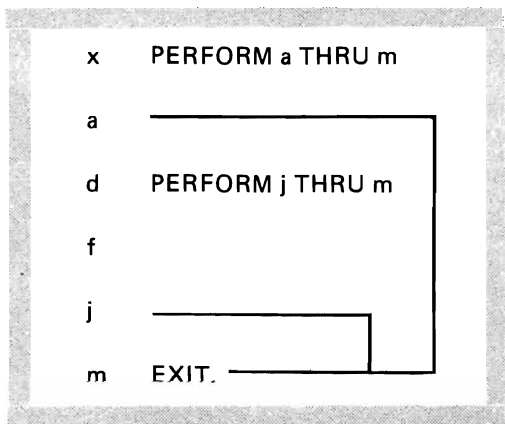
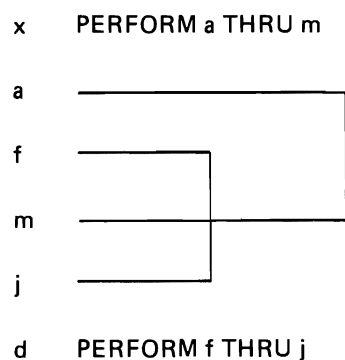
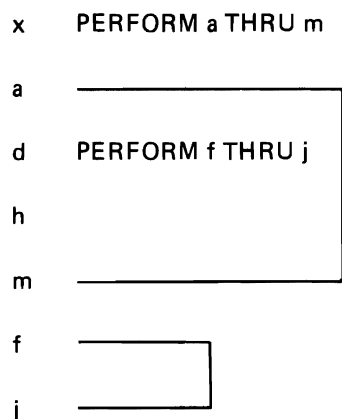
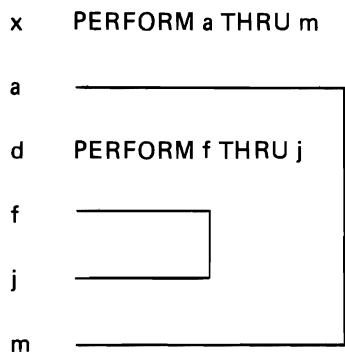


Figure 5-22. Valid PERFORM Statement Execution Sequences

Format 4

Format 4 uses the VARYING phrase. This phrase increments or decrements one or more identifiers or index-names according to the following rules. Once the condition(s) specified in the UNTIL phrase is satisfied, control is passed to the next executable statement following the PERFORM statement.

No matter how many variables are specified, the following rules apply:

- In the VARYING/AFTER phrases, when an index-name is specified:
 - The index-name is initialized and incremented or decremented according to the rules for the SET statement. For a description of the SET statement, see “Table Handling” in Chapter 6.
 - In the associated FROM phrase, an identifier must be described as an integer and have a positive value; a literal must be a positive integer.
 - In the associated BY phrase, an identifier must be described as an integer; a literal must be a nonzero integer.

- In the FROM phrase, when an index-name is specified:
 - In the associated VARYING/AFTER phrase, an identifier must be described as an integer. It is initialized as described in the SET statement.
 - In the associated BY phrase, an identifier must be described as an integer and have a nonzero value; a literal must be a nonzero integer.
- In the BY phrase, identifiers and literals must have a nonzero value.
- Changing the values of identifiers and/or index-names in the VARYING, AFTER, FROM, and BY phrases during execution changes the number of times the procedures are executed.

The way in which operands are incremented or decremented depends on the number of variables specified.

Varying One Identifier

In the following discussion, every reference to identifier-n refers equally to index-name-n except when identifier-n is the object of the BY phrase.

The following actions take place:

1. Identifier-1 is set equal to its starting value, identifier-2 or literal-2.
2. Condition-1 is evaluated:
 - a. If it is false, steps 3 through 5 are executed.
 - b. If it is true, control passes directly to the statement following the PERFORM statement.
3. Procedure-1 through procedure-2 (if specified) are executed once.
4. Identifier-1 is augmented by identifier-3 (or literal-3), and condition-1 is evaluated again.
5. Steps 2 through 4 are repeated until condition-1 is true.

Figure 5-23 is a flowchart illustrating the logic of the PERFORM statement when one identifier is varied.

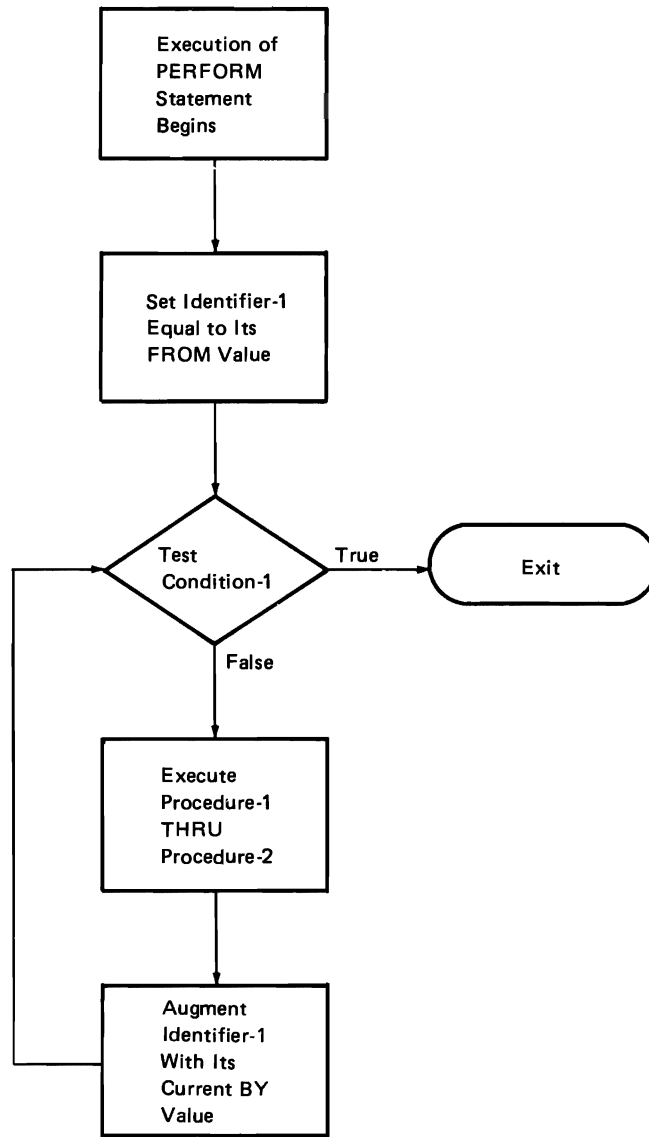


Figure 5-23. Format 4 PERFORM Statement Logic – Varying One Identifier

The following example shows a PERFORM statement varying one identifier. This PERFORM logic is executed 100 times.

```

... 1 . . . 2 . . . 3 . . . 4 . . . 5 . . . 6 . . . 7
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SUB1 PIC 999
01 TOTAL-HOLD PIC 99 VALUE 57.
01 HOLD-2 PIC 99 VALUE 10.
01 HOLD-THE-SUM PIC 99 VALUE ZERO.
01 TABLE-ELEMENT.
   03 ELEMENTS-OF-TABLE PIC 9 OCCURS 100 TIMES.
PROCEDURE DIVISION.
100-START-PROCESSING.
* THIS PERFORM LOGIC IS EXECUTED 100 TIMES.
  PERFORM SAMPLE-PERFORM THRU PERFORM-EXIT
    VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 GREATER THAN 100.
* THIS ADD STATEMENT IS EXECUTED AFTER PERFORM IS DONE.
  ADD TOTAL-HOLD HOLD-2 GIVING HOLD-THE-SUM.
  DISPLAY "TOTAL OF TWO VARIABLES = " HOLD-THE-SUM
    UPON MYTUBE.
* PERFORM ANOTHER-WAY-TO-INITIALIZE THRU AWTI-EXIT.
  THE TABLE WILL BE ALL ZEROS AND SHOULD PRINT AS SUCH.
  DISPLAY "THE TABLE " TABLE-ELEMENT UPON MYTUBE.
  STOP RUN.
SAMPLE-PERFORM.
  MOVE ZEROS TO ELEMENTS-OF-TABLE (SUB1).
PERFORM-EXIT.
  EXIT.
ANOTHER-WAY-TO-INITIALIZE.
  MOVE ZEROS TO TABLE-ELEMENT.
AWTI-EXIT.
  EXIT.

```

Varying Two Identifiers

In the following discussion, every reference to identifier-n refers equally to index-name-n except when identifier-n is the object of the BY phrase.

The following actions take place:

1. Identifier-1 and identifier-4 are set to their initial values, identifier-2 (or literal-2) and identifier-5 (or literal-5), respectively.
2. Condition-1 is evaluated:
 - a. If it is false, steps 3 through 7 are executed.
 - b. If it is true, control passes directly to the statement following the PERFORM statement.
3. Condition-2 is evaluated:
 - a. If it is false, steps 4 through 6 are executed.
 - b. If it is true, identifier-4 is set to the current value of identifier-5, and identifier-1 is augmented by identifier-3 (or literal-3), and step 2 is repeated.
4. Procedure-1 through procedure-2 (if specified) are executed once.

5. Identifier-4 is augmented by identifier-6 (or literal-6).
6. Steps 3 through 5 are repeated until condition-2 is true.
7. Steps 2 through 6 are repeated until condition-1 is true.

At the end of PERFORM statement execution, identifier-4 contains the current value of identifier-5. Identifier-1 has a value that exceeds the last used setting by the increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case identifier-1 contains the current value of identifier-2).

Figure 5-24 is a flowchart illustrating the logic of the PERFORM statement when two identifiers are varied.

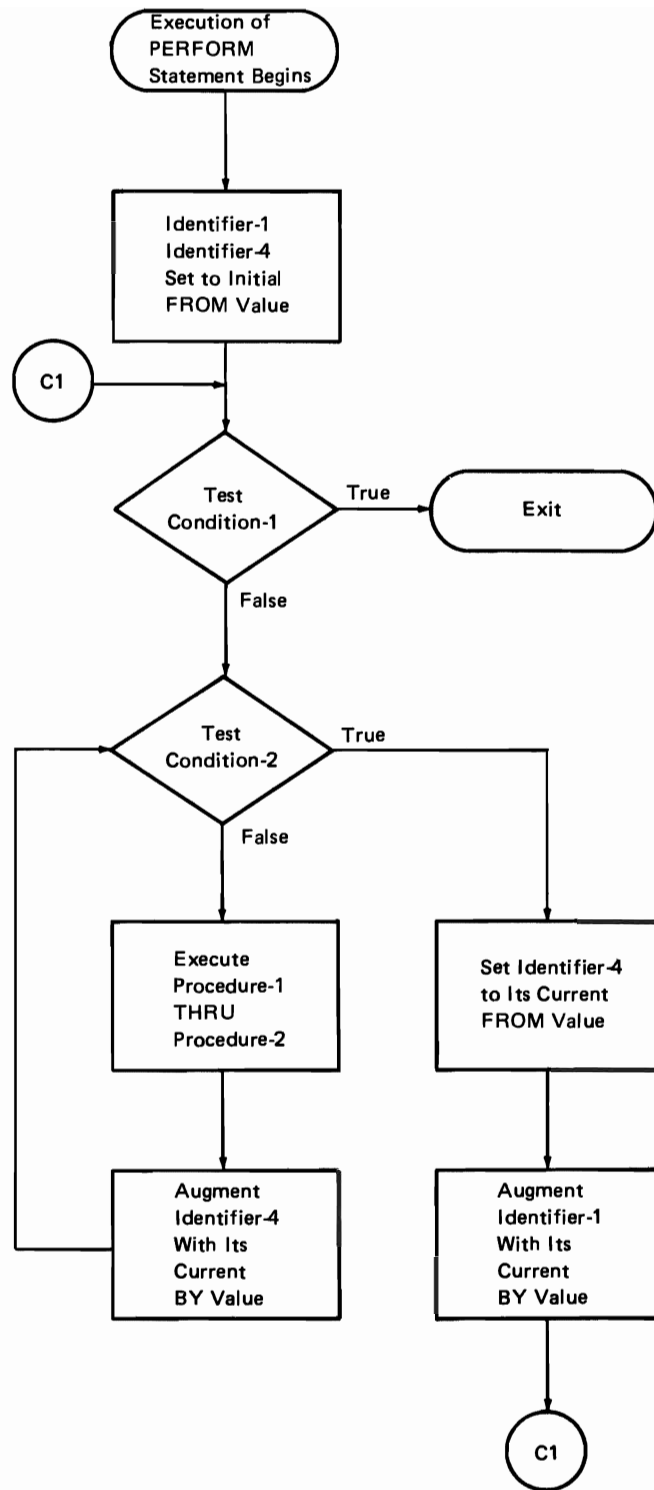


Figure 5-24. Format 4 PERFORM Statement Logic – Varying Two Identifiers

The following example shows a PERFORM statement varying two identifiers. This PERFORM logic is executed 126 times. This program searches a table and gives a total of female employees.

```

... 1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7

DATA DIVISION.
FILE SECTION.
FD PRINTED-REPORT
  LABEL RECORDS OMITTED.
01 PRINT-OUT          PIC X(132).
FD EMPLOYEE-DATA
  BLOCK CONTAINS 1 RECORDS
  RECORD CONTAINS 80 CHARACTERS
  LABEL RECORDS STANDARD
  DATA RECORD IS EMPLOYEE-RECORD.
01 EMPLOYEE-RECORD   PIC X(90).
WORKING-STORAGE SECTION.
01 RECORDS-IN        PIC 9(5)    VALUE ZEROS.
01 EOF-SW            PIC X      VALUE "N".
01 HOLD-INPUT-RECORD.
  03 EMPLOYEE-SEX    PIC 9.
    88 MALE          VALUE IS 1.
    88 FEMALE        VALUE IS 2.
  03 EMPLOYEE-RACE   PIC 9.
    88 RACE-CODES    VALUES ARE 1 THRU 7.
  03 EMPLOYEE-JOB-CLASS PIC 99.
    88 JOB-CLASS     VALUES ARE 01 THRU 18.
  03 FILLER          PIC X(76)   VALUE SPACES.
01 EMPLOYEE-TABLE.
  03 E-SEX           OCCURS 2 TIMES.
    05 E-RACE        OCCURS 7 TIMES.
    07 E-JOB         OCCURS 18 TIMES PIC 99.
01 SUB1              PIC 99.
01 SUB2              PIC 99.
01 SUB3              PIC 99.
01 TOTAL-WOMEN      PIC 9(5)    VALUE ZEROS.
PROCEDURE DIVISION.
100-START-IT.
  OPEN INPUT EMPLOYEE-DATA OUTPUT PRINTED-REPORT.
  MOVE ZEROS TO EMPLOYEE-TABLE.
200-READ-IT.
  READ EMPLOYEE-DATA RECORD INTO HOLD-INPUT-RECORD
    AT END MOVE "Y" TO EOF-SW.
  ADD 1 TO RECORDS-IN.
300-MAINLINE-LOGIC.
* THE PERFORM STATEMENT USING 2 VARIABLES WILL BE DONE 126
* TIMES
  PERFORM LOAD-TABLE UNTIL EOF-SW = "Y".
  PERFORM FIND-NUMBER-OF-WOMEN
    VARYING SUB2 FROM 1 BY 1 UNTIL SUB2 > 7
    AFTER SUB3 FROM 1 BY 1 UNTIL SUB3 > 18.
  PERFORM WRITE-REPORT THRU WR-EXIT.
  DISPLAY "TOTAL RECORDS IN " RECORDS-IN UPON MYTUBE.
  STOP RUN.
LOAD-TABLE.
  MOVE EMPLOYEE-SEX TO SUB1.
  MOVE EMPLOYEE-RACE TO SUB2.
  MOVE EMPLOYEE-JOB-CLASS TO SUB3.
  ADD 1 TO E-JOB (SUB1 SUB2 SUB3).
  PERFORM 200-READ-IT.
FIND-NUMBER-OF-WOMEN.
  ADD E-JOB (2 SUB2 SUB3) TO TOTAL-WOMEN.
WRITE-REPORT.
  MOVE TOTAL-WOMEN TO PRINT-OUT.
  WRITE PRINT-OUT.
WR-EXIT.
  EXIT.

```

Varying Three Identifiers

In the following discussion, every reference to identifier-n refers equally to index-name-n except when identifier-n is the object of the BY phrase.

The actions are the same as for varying two identifiers except that identifier-7 goes through the complete cycle each time that identifier-4 is augmented by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

At the end of PERFORM statement execution, identifier-4 and identifier-7 contain the current values of identifier-5 and identifier-8, respectively. Identifier-1 has a value exceeding its last used setting by one increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case identifier-1 contains the current value of identifier-2).

Figure 5-25 is a flowchart illustrating the logic of the PERFORM statement when three identifiers are varied.

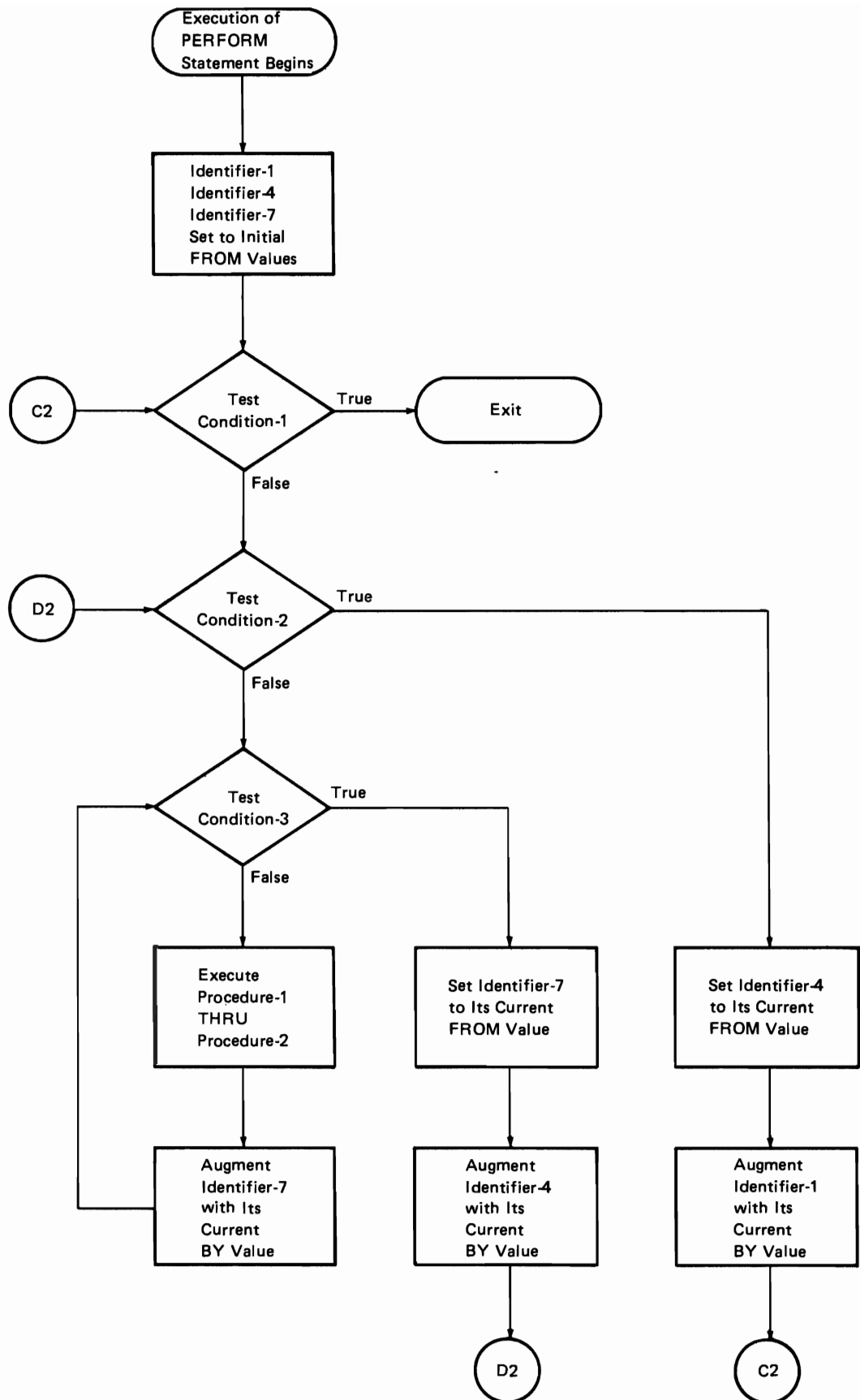


Figure 5-25. Format 4 PERFORM Statement Logic – Varying Three Identifiers

The following example shows a PERFORM statement varying three identifiers. This PERFORM logic is executed 250 times.

```

.. 1 .. 2 .. 3 .. 4 .. 5 .. 6 .. 7

DATA DIVISION.
WORKING-STORAGE SECTION.
01 SUB1 PIC 99.
01 SUB2 PIC 99.
01 SUB3 PIC 99.
01 TEST-IT PIC 99 VALUE 00.
01 TOTAL-RECS PIC 99 VALUE ZEROS.
01 COMPANY-TABLE.
   05 DIVISION-IN OCCURS 10 TIMES.
      10 DIVISION-NAME PIC X(10).
      10 DIVISION-NUMBER PIC 9(4).
      10 SECTION-IN OCCURS 5 TIMES.
         15 UNIT-IN OCCURS 5 TIMES.
            20 UNIT-NAME PIC X(5).
            20 UNIT-NUMBER PIC 9(4).
PROCEDURE DIVISION.
100-START-PROCESSING.
* THIS PERFORM LOGIC IS EXECUTED 250 TIMES
  PERFORM ZERO-OUT-BIG-TABLE
    VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 10
* SUB1 IS VARIED LAST
  AFTER SUB2 FROM 1 BY 1 UNTIL SUB2 > 5
* SUB2 IS VARIED SECOND
  AFTER SUB3 FROM 1 BY 1 UNTIL SUB3 > 5.
* SUB3 IS VARIED FIRST
  PERFORM ADDRESS-THE-VARIABLES THRU ATV-EXIT.
  DISPLAY "VARIABLE TEST-IT = " TEST-IT UPON MYTUBE.
  STOP RUN.
ZERO-OUT-BIG-TABLE.
  MOVE ZEROS TO UNIT-IN (SUB1 SUB2 SUB3).
ADDRESS-THE-VARIABLES.
  IF UNIT-NUMBER OF UNIT-IN OF SECTION-IN OF DIVISION-IN
    OF COMPANY-TABLE (3 4 5) = 0
    ADD 1 TO TEST-IT.
ATV-EXIT.
EXIT.

```

Programming Note: The procedures executed by a PERFORM statement are, in effect, a closed subroutine that can be entered from many other points in the program.

The Format 4 PERFORM statement is especially useful in table handling. One Format 4 PERFORM statement can serially search an entire 3-dimensional table.

Segmentation Information

A PERFORM statement appearing in a permanent segment can have in its range only one of the following:

- Sections, each of which has a segment number less than 50
- Sections and/or paragraphs wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have in its range only one of the following:

- Sections, each of which has a segment number less than 50

- Sections and/or paragraphs wholly contained within the same independent segment as the PERFORM statement.

Control is passed to the performed procedures only once for each execution of the PERFORM statement.

STOP Statement

The STOP statement halts the object program either temporarily or permanently.

Format

$$\underline{\text{STOP}} \left\{ \begin{array}{l} \text{RUN} \\ \text{literal} \end{array} \right\}$$

The literal can be numeric or nonnumeric, and can be any figurative constant except ALL literal. If the literal is numeric, it must be an unsigned integer.

When STOP literal is specified, the literal is communicated to the system operator for batch jobs and to the work station for interactive jobs. Program execution is suspended. Execution is resumed only after operator intervention.

The operator response determines whether the run unit continues at the next executable statement in the sequence, or a STOP RUN is executed.

Operator Response Action

G (default) Continue at next instruction.

C Terminate execution of the run unit. Escape message CBE9001 is issued to the caller of the COBOL run unit. For batch jobs, the job is canceled if the CNLSEV parameter for the job contains a value that is less than or equal to the severity of the message.

The output of the STOP literal contains the program-name followed by the literal.

If the literal cannot be contained in the length of one line of the display device, the Help key must be used to display the entire literal.

When STOP RUN is specified, execution of the run unit is terminated. If a STOP RUN statement appears in a sequence of imperative statements, it must be the last or the only statement in the sequence. All files should be closed before a STOP RUN statement is executed. If you do not close the files, they are closed by compiler generated code. An implicit STOP RUN is always generated after the last statement in the source program.

Programming Note: The STOP literal statement is useful for special situations when operator intervention is needed during program execution.

Compiler-Directing Statements

Compiler-directing statements provide instructions to the COBOL compiler. The compiler-directing statements are COPY, ENTER, and USE.

Only the ENTER statement is discussed in this chapter. The COPY statement is discussed under “Source Program Library” in Chapter 6. The USE statements are discussed under “Declaratives” in this chapter and under “Debugging Features” in Chapter 6.

ENTER Statement

The System/38 COBOL compiler does not allow another source language to be used in COBOL source programs. Therefore, the ENTER statement is not required or used by the System/38 COBOL compiler, but is only treated as a comment.

Format

ENTER language-name [routine-name] .

Language-name and routine-name can be any user-defined word. The compiler expects a valid COBOL statement to immediately follow the ENTER statement.

Chapter 6. Additional Functions

System/38 COBOL offers several additional functions that are useful to programmers who are writing more advanced applications. The additional functions provided by the System/38 COBOL discussed in this chapter are:

- Table Handling
- SORT-MERGE
- Source Program Library
- Segmentation
- Inter-Program Communication
- Debugging
- FIPS Flagger.

TABLE HANDLING

Tables are often used in data processing. A table is a set of logically consecutive items, each of which has the same data description as the other items in the set. The items in a table can be described as separate contiguous items. However, this approach may not be satisfactory for two reasons. From a documentation standpoint, the homogeneity of the data items is not apparent; secondly, repetitive coding to reference unique data-names becomes a severe problem. Thus, a method of data reference is used which makes it possible to refer to all or to part of one table as an entity.

Table Handling Concepts

In COBOL, a table is defined with an OCCURS clause in its data description. The OCCURS clause specifies that the named item is to be repeated as many times as stated. The item so named is considered a table element, and its name and description apply to each repetition (or occurrence) of the item. Because the occurrences are not given unique data-names, reference to a particular occurrence can be made only by specifying the data-name of the table element, together with the occurrence number of the desired item within the element.

The occurrence number is known as a subscript and the technique of supplying the occurrence number of individual table elements is called subscripting. A related technique, called indexing, is also available for table references. Both subscripting and indexing are described in subsequent sections.

Table Definition

COBOL allows tables in one, two or three dimensions.

To define a one-dimensional table, the user writes an OCCURS clause as part of the definition of a table element. However, the OCCURS clause must not appear in a data description entry that has a 01, 66, 77, or 88 level-number. For example:

```
01 TABLE-ONE.  
  05 ELEMENT-ONE OCCURS 3 TIMES.  
    10 ELEMENT-A PIC X(4).  
    10 ELEMENT-B PIC 9(4).
```

TABLE-ONE is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-ONE.

To define a two-dimensional table, a one-dimensional table is defined within each occurrence of another one-dimensional table. For example:

```
01 TABLE-TWO.  
  05 ELEMENT-ONE OCCURS 3 TIMES.  
    10 ELEMENT-TWO OCCURS 3 TIMES.  
      15 ELEMENT-A PIC X(4).  
      15 ELEMENT-B PIC 9(4).
```

TABLE-TWO is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-TWO.

To define a three-dimensional table, a one-dimensional table is defined within each occurrence of another one-dimensional table, which is itself contained within each occurrence of another one-dimensional table. For example:

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

TABLE-THREE is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-THREE is an element of a three-dimensional table that occurs two times within each occurrence of ELEMENT-TWO. Figure 6-1 shows the storage layout for TABLE-THREE.

ELEMENT-ONE Occurs Three Times	ELEMENT-TWO Occurs Three Times	ELEMENT-THREE Occurs Two Times	Byte Dis- placement
ELEMENT-ONE (1)	ELEMENT-TWO (1, 1)	ELEMENT-THREE (1, 1, 1)	0
		ELEMENT-THREE (1, 1, 2)	8
	ELEMENT-TWO (1, 2)	ELEMENT-THREE (1, 2, 1)	16
		ELEMENT-THREE (1, 2, 2)	24
	ELEMENT-TWO (1, 3)	ELEMENT-THREE (1, 3, 1)	32
		ELEMENT-THREE (1, 3, 2)	40
ELEMENT-ONE (2)	ELEMENT-TWO (2, 1)	ELEMENT-THREE (2, 1, 1)	48
		ELEMENT-THREE (2, 1, 2)	56
	ELEMENT-TWO (2, 2)	ELEMENT-THREE (2, 2, 1)	64
		ELEMENT-THREE (2, 2, 2)	72
	ELEMENT-TWO (2, 3)	ELEMENT-THREE (2, 3, 1)	80
		ELEMENT-THREE (2, 3, 2)	88
ELEMENT-ONE (3)	ELEMENT-TWO (3, 1)	ELEMENT-THREE (3, 1, 1)	96
		ELEMENT-THREE (3, 1, 2)	104
	ELEMENT-TWO (3, 2)	ELEMENT-THREE (3, 2, 1)	112
		ELEMENT-THREE (3, 2, 2)	120
	ELEMENT-TWO (3, 3)	ELEMENT-THREE (3, 3, 1)	128
		ELEMENT-THREE (3, 3, 2)	136
			144

Figure 6-1. Storage Layout for TABLE-THREE

Table References

Whenever the user refers to a table element, or to any item associated with a table element, the reference must indicate which occurrence is intended.

For a one-dimensional table, the occurrence number of the desired element gives the complete information. For tables of more than one dimension, an occurrence number for each dimension must be supplied. In the three-dimensional table defined in the previous discussion, for example, a reference to ELEMENT-THREE must supply the occurrence number for ELEMENT-ONE, ELEMENT-TWO, and ELEMENT-THREE. Either subscripting or indexing, described in the following paragraphs, can be used to supply the necessary references.

Subscripting

Subscripting is a method of providing table references through the use of subscripts. A subscript is an integer value that specifies the occurrence number of a table element. Subscripts can be used only when reference is made to an individual item within a table element.

Format

$$\left\{ \begin{array}{l} \text{data-name-1} \\ \text{condition-name} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right\} \text{data-name-2} \right] \dots \left(\text{subscript-1} \left[\text{, subscript-2} \left[\text{, subscript-3} \right] \right] \right)$$

Data-name-1 must be the name of a table element. (Note that when qualification is used, it is data-name-1 that is subscripted, not data-name-2.)

The subscript can be represented either by a literal or a data-name.

A literal subscript must be an integer, and it must have a value of one or greater. The literal can have a positive sign or it may be unsigned. Negative subscript values are not permitted. For example, the following are valid literal subscript references to TABLE-THREE:

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

A data-name subscript must be described as an elementary numeric integer data item. A data-name subscript may be qualified; it may not be subscripted or indexed. For example, assuming that SUB1, SUB2, and SUB3 are all items subordinate to SUBSCRIPT-ITEM, valid data-name subscript references to TABLE-THREE are:

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

The set of one to three subscripts must be written within a balanced pair of parentheses immediately following data-name-1 or its last qualifier. One or more spaces can optionally precede the opening parenthesis.

When more than one subscript is specified, each subscript must be separated from the next by either a space or a comma and a space.

When more than one subscript is required, the subscripts are written in the order of successively less inclusive data dimensions. For example, in the table reference ELEMENT-THREE (3, 2, 1), the first value (3) refers to the occurrence within ELEMENT-ONE, the second value (2) refers to the occurrence within ELEMENT-TWO, and the third value (1) refers to the occurrence within ELEMENT-THREE.

The lowest possible subscript value is 1; this value points to the first occurrence within the table element. The next sequential elements are pointed to by subscripts with values 2, 3, and so on. The highest permissible subscript value in any particular table element is the maximum number of occurrences specified in the OCCURS clause. For example, in TABLE-THREE the highest possible subscript value for ELEMENT-ONE is 3, for ELEMENT-TWO is 3, and for ELEMENT-THREE is 2.

If the RANGE option is specified or implied (see “COBOL Command Statement” or “Process Statement” in Chapter 8), the system ensures that the subscript value is valid. If the RANGE option is not active, it is your responsibility to ensure that the subscript value is valid.

Specify the RANGE option for subscripts only. The RANGE option does *not* verify that indexes are valid.

The following example shows subscripting using a 3-level table. In this example, UNIT-NUMBER does not need qualification and could also be referenced as UNIT-NUMBER (3, 4, 5).

```

1 . . . . . 2 . . . . . 3 . . . . . 4 . . . . . 5 . . . . . 6 . . . . . 7
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SUB1 PIC 99.
01 SUB2 PIC 99.
01 SUB3 PIC 99.
01 TEST-IT PIC 99 VALUE 00.
01 TOTAL-RECS PIC 99 VALUE ZEROS.
01 COMPANY-TABLE.
   05 DIVISION-IN OCCURS 10 TIMES.
      10 DIVISION-NAME PIC X(10).
      10 DIVISION-NUMBER PIC 9(4).
      10 SECTION-IN OCCURS 5 TIMES.
         15 UNIT-IN OCCURS 5 TIMES.
            20 UNIT-NAME PIC X(5).
            20 UNIT-NUMBER PIC 9(4).
PROCEDURE DIVISION.
100-START-PROCESSING.
   PERFORM ZERO-OUT-BIG-TABLE
      VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 10
*   SUB1 IS VARIED LAST
      AFTER SUB2 FROM 1 BY 1 UNTIL SUB2 > 5
*   SUB2 IS VARIED SECOND
      AFTER SUB3 FROM 1 BY 1 UNTIL SUB3 > 5.
*   SUB3 IS VARIED FIRST
   PERFORM ADDRESS-THE-VARIABLES THRU ATV-EXIT.
   DISPLAY "VARIABLE TEST-IT = " TEST-IT UPON MYTUBE.
   STOP-RUN.
ZERO-OUT-BIG-TABLE.
   MOVE ZEROS TO UNIT-IN (SUB1 SUB2 SUB3).
ADDRESS-THE-VARIABLES.
   IF UNIT-NUMBER OF UNIT-IN OF SECTION-IN OF DIVISION-IN
      OF COMPANY-TABLE (3 4 5) = 0
      ADD 1 TO TEST-IT.
ATV-EXIT.
EXIT.

```

Subscripting {

Indexing

Indexing is the method of providing table references through the use of indexes. An index is a compiler-generated storage area used to store table element occurrence numbers. For System/38 COBOL, the index contains a value that is an offset into the table.

Format

$$\left\{ \begin{array}{l} \text{data-name-1} \\ \text{condition-name} \end{array} \right\} \left[\left[\begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right] \text{data-name-2} \right] \dots \left(\left\{ \begin{array}{l} \text{index-name-1} \left[\left[\pm \right] \text{literal-2} \right] \\ \text{literal-1} \end{array} \right\} \right. \\ \left. \left[\left\{ \begin{array}{l} \text{index-name-2} \left[\left[\pm \right] \text{literal-4} \right] \right\} \left[\left\{ \begin{array}{l} \text{index-name-3} \left[\left[\pm \right] \text{literal-6} \right] \right\} \right] \right] \right] \right)$$

Data-name-1 must be the name of a table element. (Note that when qualification is used, it is data-name-1 that is indexed rather than data-name-2.)

Each index-name identifies an index to be used in table references. The index-name is specified through the INDEXED BY phrase in the OCCURS clause.

Each index named is a compiler-generated storage area, 2 bytes in length. Two forms of indexing are provided: direct and relative.

In direct indexing, the index-name is in the form of a subscript. In relative indexing, the index-name is followed by a space, a + or a -, another space, and an unsigned numeric literal. The literal is considered to be an occurrence number, and is converted to an index value before being added to or subtracted from the index-name index.

To be valid during execution, an index value must correspond to a table element occurrence number that is not less than one, or greater than the highest permissible occurrence number. This restriction applies to both direct and relative indexing.

The RANGE option (see “COBOL Command Statement” or “Process Statement” in Chapter 8) does *not* cause the system to verify that index values are valid. It is your responsibility to ensure valid index values.

An index-name must be initialized through a SET, PERFORM-Format 4, or SEARCH ALL statement before it is used in a table reference.

One or more index references (direct or relative) can be specified together with literal subscripts.

Further information on index-names is given later in this chapter in the description of the INDEXED BY phrase of the OCCURS clause.

Restrictions on Subscripting and Indexing

- A data-name must not be subscripted or indexed when it is being used as a subscript or qualifier.
- Indexing is not permitted when subscripting is not permitted.
- An index can be modified only by a PERFORM, SEARCH, or SET statement.
- When a literal is used in a subscript, it must be a positive or unsigned integer.
- When a literal is used in relative indexing, it must be an unsigned integer.

Table Initialization

A table can contain static values or dynamic values. Static values remain the same through every execution of the object program. When this is true, the initial values of table elements can be specified in Working-Storage in one of two ways:

- The table can be described as a record containing contiguous subordinate data description entries, each of which contains a VALUE clause for the initial value. The record is then redescribed through a REDEFINES entry that contains a subordinate entry with an OCCURS

clause. Because of the OCCURS clause, the subordinate entries of the redefined entry are repeated. For example:

```
01 TABLE-ONE.  
   05 ELEMENT-ONE PICTURE X VALUE "1".  
   05 ELEMENT-TWO PICTURE X VALUE "2".  
   05 ELEMENT-THREE PICTURE X VALUE "3".  
   05 ELEMENT-FOUR PICTURE X VALUE "4".  
01 TABLE-TWO REDEFINES TABLE-ONE.  
   05 OCCURS-ELEMENT OCCURS 4 TIMES  
     PICTURE X.
```

- If the subordinate entries do not require separate handling, the VALUE of the entire entry can be given in the entry that names the table. The lower level entries then contain OCCURS clauses, and show the hierarchical structure of the table. The subordinate entries must not contain VALUE clauses. For example:

```
01 TABLE-ONE VALUE "1234".  
   05 TABLE-TWO OCCURS 4 TIMES  
     PICTURE X.
```

Dynamic values may change during one execution of the object program, or from one execution to another. If the dynamic values are always the same at the beginning of object program execution, they can be initialized in the same manner as static values. If the initial values change from one execution to the next, then the table can be defined without initial values, and the changed values can be placed in the table before any table reference is made.

Tables can be initialized to a common value by a MOVE to the group item that defines the entire table. For example:

```
MOVE SPACES TO TABLE-ONE.
```

However, care should be exercised when this method is used with a table containing non-display type elements. For example:

```
01 BINARY-TABLE.  
   05 BINARY-COUNT OCCURS 4 TIMES  
     PIC 9999 COMP-4.
```

```
MOVE ZEROS TO BINARY-TABLE.
```

The MOVE statement does not fill BINARY-TABLE with binary zeros, but with display-type zeros, hex "FO".

The following example shows two ways of initializing a table with zeros:

```

1 . . . . . 2 . . . . . 3 . . . . . 4 . . . . . 5 . . . . . 6 . . . . . 7
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SUB1 PIC 999.
01 TABLE-OF-ELEMENTS.
03 ELEMENTS-OF-TABLE PIC 9 OCCURS 100 TIMES.
PROCEDURE DIVISION.
100-START-PROCESSING.
PERFORM SAMPLE-PERFORM THRU PERFORM-EXIT
VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 GREATER THAN 100.
PERFORM ANOTHER-WAY-TO-INITIALIZE THRU AWTI-EXIT.
* THE TABLE WILL BE ALL ZEROS AND SHOULD PRINT AS SUCH.
DISPLAY "THE TABLE " TABLE-OF-ELEMENTS UPON MYTUBE.
STOP-RUN.
{
SAMPLE-PERFORM.
MOVE ZEROS TO ELEMENTS-OF-TABLE (SUB1).
PERFORM-EXIT.
EXIT.
}
{
ANOTHER-WAY-TO-INITIALIZE.
MOVE ZEROS TO TABLE-OF-ELEMENTS.
AWTI-EXIT.
EXIT.
}

```

Initializing
Table to
Zeros

Data Division – Table Handling

COBOL Data Division clauses used for Table Handling are the OCCURS clause and the USAGE IS INDEX clause.

OCCURS Clause

The OCCURS clause eliminates the need to specify separate entries for repeated data items; it also supplies the information necessary for the use of subscripts or indexes. The formats of the OCCURS clause are as follows:

Format 1 – Fixed Length Tables

OCCURS integer-2 TIMES

[{ ASCENDING } KEY IS data-name-2 [data-name-3] ...] ...
[DESCENDING] ...]
[INDEXED BY index-name-1 [index-name-2] ...]

Format 2 – Variable Length Tables

OCCURS integer-1 TO integer-2 TIMES

DEPENDING ON data-name-1

[{ ASCENDING
DESCENDING } KEY IS data-name-2 [data-name-3] ...] ...
[INDEXED BY index-name-1 [index-name-2] ...]

The subject of an OCCURS clause is the data-name of the data item containing the OCCURS clause. Except for the OCCURS clause itself, data description clauses used with the subject apply to each occurrence of the item described.

Whenever the subject is used in any statement – other than SEARCH or USE FOR DEBUGGING, or unless it is the object of a REDEFINES clause – the subject must be subscripted or indexed. When it is subscripted or indexed, the subject refers to one occurrence within the table element.

Whenever the subject is used in a SEARCH or USE FOR DEBUGGING statement, or when it is the object of a REDEFINES clause, the subject must not be subscripted or indexed. When it is not subscripted or indexed, the subject represents the entire table length.

The table must contain less than 32 768 occurrences, the length of a table element must be less than 32 K bytes, and the length of the whole table must be less than 32 K bytes.

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

All integers must be positive nonzero integers.

The OCCURS clause cannot be specified in a data description entry that:

- Has a level-01, level-66, level-77, or level-88 number.
- Describes an item of variable size (an item is of variable size if any subordinate entry contains an OCCURS DEPENDING ON clause).
- Describes redefined data items. (However, a redefined item can be subordinate to an item containing an OCCURS clause.) See “REDEFINES Clause” in Chapter 4.

Fixed Length Tables

When Format 1 is used, integer-2 specifies the exact number of occurrences.

Integer-2 must be greater than zero and less than 32 768.

Because three subscripts or indexes are allowed, three nested levels of the Format 1 OCCURS clause are allowed.

Variable Length Tables

When the OCCURS DEPENDING ON clause is specified, integer-1 represents the minimum number of occurrences, and integer-2 represents the maximum number of occurrences. The value of integer-1 must be one or greater; it must also be less than integer-2. Integer-2 must be less than 32 768. The length of the subject item is fixed; it is only the number of repetitions of the subject item that is variable.

Data-name-1 is the object of the OCCURS DEPENDING ON clause. The object is the data item whose current value represents the current number of occurrences of the subject item. The object of the OCCURS DEPENDING ON clause:

- Must be described as a positive integer. That is, if data-name-1 is described as a signed item, at execution time it must contain positive data.
- Must not occupy any storage position within the range of this table. That is, the object must not occupy any storage position from the first character position in this table through the last character position in this record description entry.
- Must contain a value within the range of integer-1 and integer-2, inclusive.

The value of the object of the OCCURS DEPENDING ON clause specifies that part of the table element available to the object program. Items whose occurrence numbers exceed the value of the object are not available. If, during execution, the value of the object is reduced, the contents of items whose occurrence numbers exceed the new value of the object are unpredictable.

When a group item containing a subordinate OCCURS DEPENDING ON item is referred to, the current value of the object determines which part of the table area is used in the operation.

In one record description entry, any entry that contains an OCCURS DEPENDING ON clause may be followed only by items subordinate to it. The OCCURS DEPENDING ON clause cannot be specified as subordinate to another OCCURS clause. However, the Format 1 OCCURS clause may be specified as subordinate to the OCCURS DEPENDING ON clause; in this case, a table of up to three dimensions may be specified.

ASCENDING/DESCENDING KEY Phrase

The ASCENDING/DESCENDING KEY phrase specifies that the repeated data is arranged in ascending or descending key sequence (depending on the keyword specified) according to the values contained in data-name-2, data-name-3, and so on. The data-names are listed in their descending order of significance. The ASCENDING/DESCENDING KEY data items are used by the SEARCH ALL statement for a search of the table element.

The order is determined by the rules for comparison of operands. (See "Simple Conditions, Relation Condition" in Chapter 5.)

Data-name-2 must be the name of the subject entry or the name of an entry subordinate to the subject entry. If data-name-2 names the subject entry, that entire entry becomes the ASCENDING/DESCENDING KEY and is the only key that can be specified for this table element. If data-name-2 does not name the subject entry, then data-name-2, data-name-3, and so on:

- Must be subordinate to the subject of the table entry itself
- Must not be subordinate to any other entry that contains an OCCURS clause
- Must not themselves contain an OCCURS clause.

The following example illustrates the specification of ASCENDING/DESCENDING KEY data items:

```
WORKING-STORAGE SECTION.  
01 CURRENT-WEEK PICTURE 99.  
01 TABLE-RECORD.  
    05 EMPLOYEE-TABLE OCCURS 100 TIMES  
        ASCENDING KEY IS WAGE-RATE  
        EMPLOYEE-NO INDEXED BY A, B  
        10 EMPLOYEE-NAME PIC X(20)  
        10 EMPLOYEE-NO PIC 9(6).  
        10 WAGE-RATE PIC 9999V99.  
    10 WEEK-RECORD OCCURS 52 TIMES  
        ASCENDING KEY IS WEEK-NO  
        INDEXED BY C.  
        15 WEEK-NO PIC 99  
        15 AUTHORIZED-ABSENCES PIC 9.  
        15 UNAUTHORIZED-ABSENCES PIC 9.  
        15 LATENESSES PIC 9.
```

The keys for EMPLOYEE-TABLE are subordinate to that entry, and the key for WEEK-RECORD is subordinate to that subordinate entry.

When the ASCENDING/DESCENDING KEY phrase is specified, the following rules apply:

- Keys must be listed in decreasing order of significance.

- A key can have USAGE DISPLAY or COMPUTATIONAL.

IBM Extension

A key can have USAGE COMPUTATIONAL-3 or COMPUTATIONAL-4.

End of IBM Extension

- The user is responsible for ensuring that the data present in the table is arranged in ascending or descending key sequence according to the collating sequence in use.

In the preceding example, records in EMPLOYEE-TABLE must be arranged in ascending order of WAGE-RATE and in ascending order of EMPLOYEE-NO within WAGE-RATE. Records in Week-Record must be arranged in ascending order of WEEK-NO. If they are not, SEARCH ALL statement results will be unpredictable.

INDEXED BY Phrase

The INDEXED BY phrase specifies the indexes that can be used with this table element. The INDEXED BY phrase is required if indexing is used to refer to this table element.

Each index-name must follow the rules for formation of a user-defined word; at least one character must be alphabetic. Each index-name specifies an index to be created by the compiler for use by the program. These index-names are not data-names and are not identified elsewhere in the COBOL program; instead, they can be regarded as compiler generated registers for the use of this object program only. Therefore, they are not data or part of any data hierarchy; as such, each must be unique. An INDEX-NAME can only be referenced by a PERFORM, SET, or SEARCH statement, as a parameter in the USING phrase in a CALL statement, or in a relational condition comparison.

USAGE IS INDEX Clause

The USAGE IS INDEX clause specifies that the data item named has an index format. Such an item is an index data item.

Format

[USAGE IS] INDEX

An index data item is a 2-byte elementary item that can be used to save index-name values for future reference. Through the SET statement, an index data item can be assigned an index-name value. The index-name value corresponds to the displacement for an occurrence number in the table, that is (occurrence-number - 1) * entry length.

An index data item can be referred to directly only in a SEARCH statement, a SET statement, a relation condition, the USING phrase of the Procedure Division header, or the USING phrase of the CALL statement. An index data item can be part of a group item referred to in a MOVE statement or an input/output statement.

An index data item saves binary values that represent a table occurrence number; however, it is not itself necessarily defined as part of any table. Thus, when it is referenced directly in a SEARCH or SET statement, or indirectly in a MOVE or input/output statement, there is no conversion of values when the statement is executed.

The USAGE IS INDEX clause may be written at any level. If a group item is described with the USAGE IS INDEX clause, it is the elementary items within the group that are index data items; the group itself is not an index data item, and the group name cannot be used in SEARCH and SET statements or in relation conditions. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An index data item cannot be a conditional variable; it cannot have a subordinate level-88 item.

The SYNCHRONIZED, JUSTIFIED, PICTURE, BLANK WHEN ZERO, or VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

Since the format of an index data item is implementation dependent, an index data item should not be defined in the File Section.

Procedure Division – Table Handling

In the Procedure Division, the SEARCH and SET statements can be specified with indexed tables. There are also special rules involving table handling elements when they are used in relation conditions.

Relation Conditions

Comparisons involving index-names and/or index data items conform to the following rules:

- The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.
- In the comparison of an index-name with a data item (other than an index data item) or in the comparison of an index-name with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.
- In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion.

Results of any other comparison involving an index data item are undefined.

Figure 6-2 shows permissible comparisons for index-names and index data items.

First Operand	Second Operand			
	Index-name	Index Data Item	Data-Name (numeric integer only)	Numeric Literal (integer only)
Index-Name	Compare occurrence numbers	Compare without conversion	Compare occurrence number with data-name	Compare occurrence number with literal
Index Data Item	Compare without conversion	Compare without conversion	Invalid	Invalid
Data-Name (numeric integer only)	Compare occurrence number with data-name	Invalid	Invalid	Invalid
Numeric Literal (integer only)	Compare occurrence number with literal	Invalid	Invalid	Invalid

Figure 6-2. Permissible Comparisons for Index-Names and Index Data Items

SEARCH Statement

The SEARCH statement searches a table for an element that satisfies the specified condition, and adjusts the associated index to indicate that element. The formats for the SEARCH statement are:

Format 1

SEARCH identifier-1 [VARYING { identifier-2 }
 { index-name-1 }] [AT END imperative-statement-1]
 WHEN condition-1 { imperative-statement-2 }
 { NEXT SENTENCE }
 [WHEN condition-2 { imperative-statement-3 }
 { NEXT SENTENCE }] . . .

Format 2

SEARCH ALL identifier-1 [AT END imperative-statement-1]
 WHEN { data-name-1 { IS EQUAL TO } { identifier-3 }
 { IS = { literal-1 }
 { arithmetic-expression-1 } }
 { condition-name-1 }
 [AND { data-name-2 { IS EQUAL TO } { identifier-4 }
 { IS = { literal-2 }
 { arithmetic-expression-2 } }
 { condition-name-2 }] . . .
 { imperative-statement-2 }
 { NEXT SENTENCE }

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY phrase.

When specified in the SEARCH statement, identifier-1 must refer to all occurrences within the table element; it must not be subscripted or indexed.

Identifier-1 can be a data item subordinate to a data item that contains an OCCURS clause; it can be a part of a two- or three-dimensional table. In this case, the data description entry must specify an INDEXED BY phrase for each dimension of the table.

SEARCH statement execution modifies only the value in the index-name associated with identifier-1 (and, if present, of index-name-1 or identifier-2). Therefore, to search an entire two- or three-dimensional table, a SEARCH statement must be executed for each dimension. Before each execution, SET statements must be executed to reinitialize the associated index-names.

In the AT END and WHEN phrases, control passes to the next sentence after the imperative-statement is executed if any of the specified imperative-statements do not end with a GO TO statement.

Format 1

Format 1 SEARCH statement execution causes a serial search to be executed, beginning at the current index setting.

If the value of the index-name associated with identifier-1 is not greater than the highest possible occurrence number, when the search begins the following actions take place:

1. The conditions in the WHEN phrases are evaluated in the order they are written.
2. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to correspond to the next table element, and step 1 is repeated.
3. If upon evaluation, one of the WHEN conditions is satisfied, the search terminates immediately, and the imperative-statement associated with that condition is executed. The index-name identifies the table element that satisfied the condition.
4. If the end of the table is reached (that is, the incremented index-name value is greater than the highest possible occurrence number) without the WHEN condition begin satisfied, the search terminates as described in the next paragraph.

If, when the search begins, the value of the index-name associated with identifier-1 is greater than the highest possible occurrence number, the search immediately ends, and, if specified, the AT END imperative-statement is executed. If the AT END phrase is omitted, control passes to the next sentence.

Each WHEN phrase condition can be any condition as described under “Conditional Expressions” in Chapter 5.

VARYING Index-Name-1 Phrase: When the VARYING index-name-1 phrase is omitted, the first (or only) index-name for identifier-1 is used for the search. When the VARYING index-name-1 phrase is specified, one of the following actions takes place:

- If index-name-1 is an index for identifier-1, this index is used for the search. Otherwise, the first (or only) index-name is used.
- If index-name-1 is an index for another table element, then the first (or only) index-name for identifier-1 is used for the search; the occurrence number represented by index-name-1 is incremented by the same amount as the search index-name and at the same time.

VARYING Identifier-2 Phrase: When this phrase is specified, the first (or only) index-name for identifier-1 is used for the search.

Identifier-2 must be either an index data item or an elementary integer item. During the search, one of the following actions takes place:

- If identifier-2 is an index data item, then whenever the search index is incremented, the specified index item is simultaneously incremented by the same amount.
- If identifier-2 is an elementary integer item, then whenever the search index is incremented, the specified data item is simultaneously incremented by 1.

Figure 6-3 is a flowchart of a Format 1 SEARCH operation containing two WHEN phrases.

Format 2

Format 2 SEARCH ALL statement execution causes a binary search to be executed. The search index need not be initialized by SET statements, because its setting is varied during the search operation. The index used is always the index that is associated with the first index-name specified in the OCCURS clause.

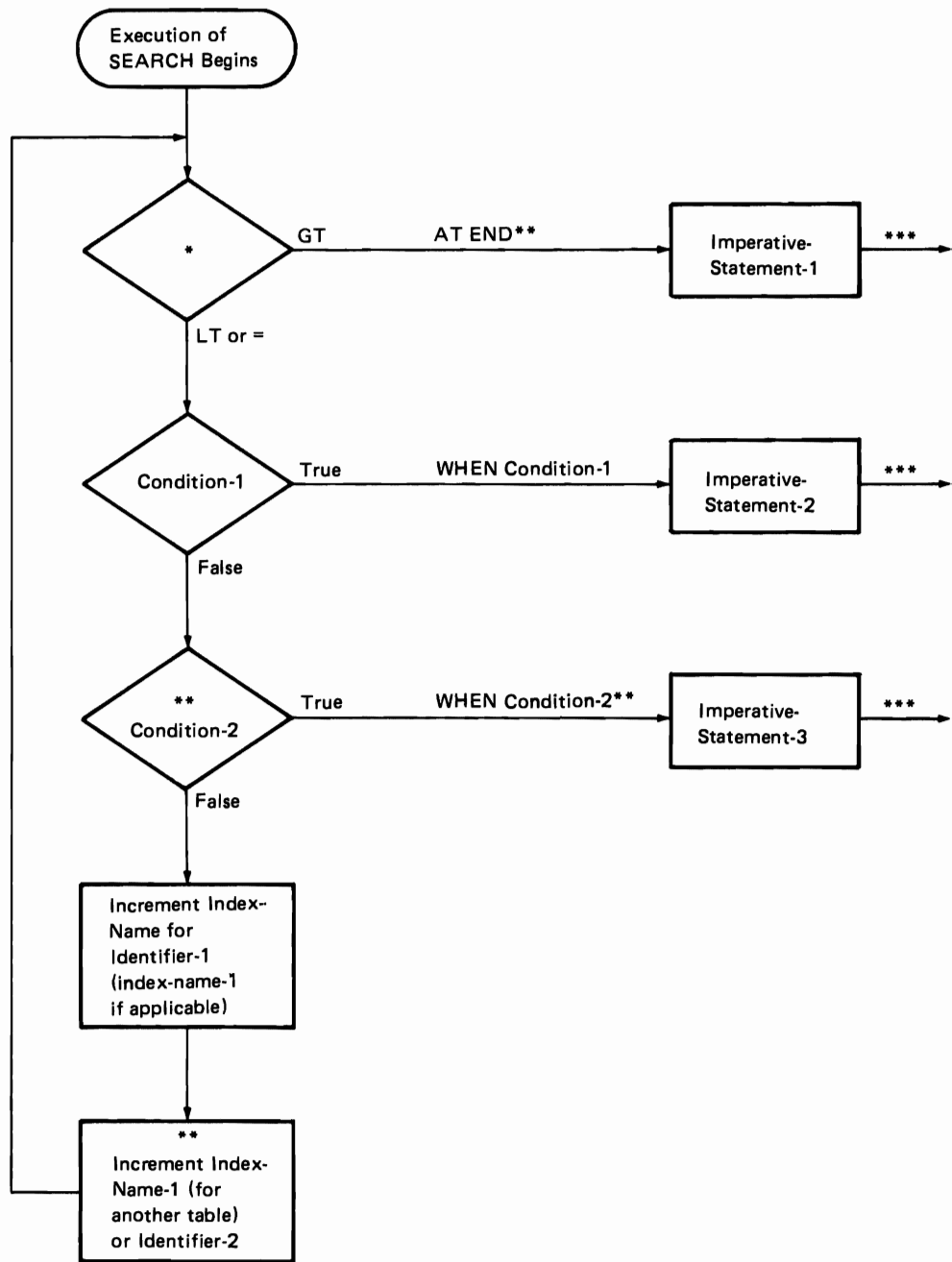
If the WHEN phrase cannot be satisfied for any setting of the index within this range, the search is unsuccessful. If the AT END phrase is specified, the AT END imperative-statement is executed. If the AT END phrase is not specified, control is passed to the next sentence. In either case, the final setting of the index is not predictable.

If the WHEN phrase can be satisfied, control passes to imperative-statement-2 and the index contains a value indicating an occurrence that allows the WHEN condition(s) to be satisfied.

WHEN Condition-Name Phrase: If the WHEN condition-name phrase is specified, each condition-name specified must have only a single value, and each must be associated with an ASCENDING/DESCENDING KEY identifier for this table element.

WHEN Relation-Condition Phrase: If WHEN relation-condition is specified, the following considerations apply:

- Data-name-1 or data-name-2 must specify an ASCENDING/DESCENDING KEY data item in the identifier-1 table element and must be indexed by the first identifier-1 index-name, along with other indexes or literals as required. Each data-name may be qualified.
- Identifier-3 and identifier-4 must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.
- Literal-1 or literal-2 must be a positive or unsigned numeric integer.



- * Index setting equals highest permissible occurrence number.
- ** These operations are included only when called for in the statement.
- *** Each of these control transfers is to the next sentence unless the imperative-statement ends with a GO TO statement.

Figure 6-3. Format 1 SEARCH with Two WHEN Phrases

- Arithmetic-expression-1 or arithmetic-expression-2 may be any of those defined under “Arithmetic Expressions” in Chapter 5 with the following restriction: any identifier in the arithmetic-expression must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.
- When an ASCENDING/DESCENDING KEY data item is specified either explicitly or implicitly in the WHEN phrase, then all preceding ASCENDING/DESCENDING KEY data-names for identifier-1 must also be specified.

The results of a SEARCH ALL operation are predictable only when both of the following apply:

- The data in the table is ordered in ascending or descending key sequence.
- The contents of the ASCENDING/DESCENDING keys specified in the WHEN phrase provide a unique table reference.

Programming Notes

Index data items cannot be used as subscripts or indexes, because of the restrictions on direct reference to them. The use of a direct indexing reference together with a relative indexing reference for the same index-name allows reference to two different occurrences of a table element for comparison purposes.

When the object of the VARYING phrase is an index-name for another table element, one Format 1 SEARCH statement looks at two table elements at once.

One Format 4 PERFORM statement can search an entire *multidimensional* table.

To ensure correct execution of a PERFORM or SEARCH statement for a variable length table, the user must make sure that the object of the OCCURS DEPENDING ON clause (data-name-1) contains a value that correctly specifies the current length of the table.

SEARCH Example

The following example searches an inventory table for items that match those from input data. The key is ITEM-NUMBER.

1 2 3 4 5 6 7

```
DATA DIVISION.
FILE SECTION.
FD SALES-DATA
  BLOCK CONTAINS 1 RECORDS
  RECORD CONTAINS 80 CHARACTERS
  LABEL RECORDS STANDARD
  DATA RECORD IS SALES-REPORTS.
01 SALES-REPORTS          PIC X(80).
FD PRINTED-REPORT
  BLOCK CONTAINS 1 RECORDS
  RECORD CONTAINS 132 CHARACTERS
  LABEL RECORDS OMITTED
  DATA RECORD IS PRINTER-OUTPUT.
01 PRINTER-OUTPUT        PIC X(132).
FD INVENTORY-DATA
  BLOCK CONTAINS 1 RECORDS
  RECORD CONTAINS 40 CHARACTERS
  LABEL RECORDS STANDARD
  DATA RECORD IS INVENTORY-RECORD.
01 INVENTORY-RECORD.
  03 I-NUMBER             PIC 9(4).
  03 INV-ID               PIC X(26).
  03 I-COST               PIC 9(8)V99.
WORKING-STORAGE SECTION.
01 EOF-SW                PIC X          VALUE "N".
01 EOF-SW2               PIC X          VALUE "N".
01 SUB1                  PIC 99.
01 RECORDS-NOT-FOUND     PIC 9(5)       VALUE ZEROS.
01 TOTAL-COSTS           PIC 9(10)      VALUE ZEROS.
01 HOLD-INPUT-DATA.
  03 INVENTORY-NUMBER    PIC 9999.
  03 PURCHASE-COST       PIC 9(4)V99.
  03 PURCHASE-DATE       PIC 9(6).
  03 FILLER              PIC X(64).
01 PRINTER-SPECS.
  03 PRINT-LINE.
    05 OUTPUT-ITEM-NUMBER PIC ZZZ9.
    05 FILLER             PIC X(48) VALUE SPACES.
    05 TOTAL-COSTS-0      PIC $(8).99.
01 PRODUCT-TABLE.
  05 INVENTORY-NUMBERS   OCCURS 50 TIMES
                        ASCENDING KEY ITEM-NUMBER
                        INDEXED BY INDEX-1.
    07 ITEM-NUMBER        PIC 9(4).
    07 ITEM-DESCRIPTION   PIC X(26).
    07 ITEM-COST          PIC 9(8)V99.
```

```

1 . . . . . 2 . . . . . 3 . . . . . 4 . . . . . 5 . . . . . 6 . . . . . 7
PROCEDURE DIVISION.
100-START-IT.
  OPEN INPUT SALES-DATA INVENTORY-DATA OUTPUT PRINTED-REPORT.
  MOVE HIGH-VALUES TO PRODUCT-TABLE.
  PERFORM READ-INVENTORY-DATA.
LOAD-TABLE-ROUTINE.
  PERFORM LOAD-IT VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 50
    OR EOF-SW2 = "Y".
  PERFORM 110-READ-IT.
200-MAIN-ROUTINE.
  PERFORM PROCESS-DATA UNTIL EOF-SW = "Y".
  MOVE TOTAL-COSTS TO TOTAL-COSTS-0.
  PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
  DISPLAY "RECORDS NOT FOUND - " RECORDS-NOT-FOUND
    UPON MYTUBE.
  STOP RUN.
PROCESS-DATA.
  SEARCH ALL INVENTORY-NUMBERS
    AT END PERFORM KEY-NOT-FOUND THRU NOT-FOUND-EXIT
    WHEN ITEM-NUMBER (INDEX-1) = INVENTORY-NUMBER
      MOVE ITEM-NUMBER (INDEX-1) TO OUTPUT-ITEM-NUMBER
      MOVE ITEM-COST (INDEX-1) TO TOTAL-COSTS-0
      ADD ITEM-COST (INDEX-1) TO TOTAL-COSTS
      PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
  PERFORM 110-READ-IT.
KEY-NOT-FOUND.
  ADD 1 TO RECORDS-NOT-FOUND.
NOT-FOUND-EXIT.
  EXIT.
LOAD-IT.
  MOVE INVENTORY-RECORD TO INVENTORY-NUMBERS (SUB1).
  PERFORM READ-INVENTORY-DATA.
WRITE-REPORT.
  WRITE PRINTER-OUTPUT FROM PRINTER-SPECS.
WRITE-REPORT-EXIT.
  EXIT.
*****END OF SAMPLE SEARCH PROGRAM*****
READ-INVENTORY-DATA.
  READ INVENTORY-DATA
    AT END MOVE "Y" TO EOF-SW2.
110-READ-IT.
  READ SALES-DATA INTO HOLD-INPUT-DATA
    AT END MOVE "Y" TO EOF-SW.

```

SET Statement

The SET statement establishes reference points for table handling operations by setting index-names to values associated with table elements. The SET statement may be used to transfer values between index-names and other elementary data items. The formats of the SET statement when it is used for table handling are described here. For information on the other formats allowed for SET statements, see "SET Statement" in Chapter 5.

Format 3

$$\underline{\text{SET}} \left\{ \begin{array}{l} \text{identifier-1} [, \text{identifier-2}] \\ \text{index-name-1} [, \text{index-name-2}] \end{array} \dots \right\} \underline{\text{TO}} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{index-name-3} \\ \text{integer-1} \end{array} \right\}$$

Format 4

$$\underline{\text{SET}} \text{ index-name-4} [, \text{index-name-5}] \dots \left\{ \begin{array}{l} \underline{\text{UP BY}} \\ \underline{\text{DOWN BY}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{integer-2} \end{array} \right\}$$

Index-names are related to a given table through the INDEXED BY phrase of the OCCURS clause; they are not further defined in the program.

When the sending and receiving fields in a SET statement share part of their storage (that is, the operands overlap), the result of the execution of such a SET statement is undefined.

Format 3

When this form of the SET statement is executed, the value of the sending field replaces (with or without conversion) the current value of the receiving field.

The receiving field can be specified as follows:

- Index-name-1, index-name-2, and so on.
- Identifier-1, identifier-2, and so on. The identifiers must name either index data items or elementary numeric integer items.

The sending field can be specified as follows:

- Identifier-3, which must name either an index data item or an elementary numeric integer item
- Index-name-3, whose value before the “SET” statement is executed must correspond to an occurrence number of its associated table
- Integer-1, which must be a positive integer.

Figure 6-4 shows valid combinations of sending and receiving fields in a Format 3 “SET” statement.

Execution of the Format 3 “SET” statement depends upon the type of receiving field, as follows:

- Index-name receiving fields (index-name-1, index-name-2, and so on) with one exception are converted to a displacement value representing the occurrence number indicated by the sending field. To be valid, the resulting index-name value must correspond to an occurrence number in its associated table element. For the one exception, when the sending field is an index data item, the value in the index data item is placed in the index-name without change.
- Index data item receiving fields (identifier-1, identifier-2, and so on) are set equal to the contents of the sending field (which must be either an index-name or an index data item); no conversion takes place. A numeric integer or literal sending field must not be specified.
- Integer data item receiving fields (identifier-1, identifier-2, and so on) are set to the occurrence number associated with the sending field, which must be an index-name. An integer data item, an index data item, or a literal sending field must not be specified.

Receiving fields are acted upon in the left-to-right order they are specified. Any subscripting or indexing associated with an identifier receiving field is evaluated immediately before the field is acted upon.

The value used for the sending field is its value at the beginning of SET statement execution.

The value for an index-name after execution of a SEARCH or PERFORM statement can be undefined; therefore, a Format 3 SET statement should be used to reinitialize such index-names before other table handling operations are attempted.

Format 4

When this form of the SET statement is executed, the value of the receiving field is incremented (UP BY) or decremented (DOWN BY) by the value in the sending field.

The receiving field can be specified by index-name-4, index-name-5, and so on. These index-name values must correspond to an occurrence number in the associated table both before and after the SET statement execution.

The sending field can be specified as identifier-4, which must be an elementary integer data item, or as integer-2, which must be an integer.

When the Format 4 SET statement is executed, the contents of the receiving field are incremented (UP BY) or decremented (DOWN BY) by the value of identifier-4 or integer-2. Receiving fields are acted upon in the left-to-right order they are specified. The value of the sending field at the beginning of SET statement execution is used for all receiving fields.

Sending Field	Receiving Field		
	Index-Name	Index Data Item	Integer Data Item
Index-name	Valid	Valid ¹	Valid
Index data item	Valid ¹	Valid	
Integer data item	Valid		
Integer literal	Valid		

¹No conversion takes place.

Figure 6-4. Sending and Receiving Fields for Format 3 SET Statements

SORT/MERGE

Arranging records in a particular order or sequence is a common requirement in data processing; such record ordering can be accomplished using sort or merge operations. While both operations accomplish record ordering, the functions and capabilities of a sort and a merge are different.

A sort produces an ordered file from one or more input files that can be completely unordered as to sort sequence. Thus, the sort operation must accept unordered input and produce ordered output.

A merge produces an ordered file from two or more input files, each of which is already ordered in the merge sequence.

IBM Extension

Input files need not be sequenced prior to a merge operation.

End of IBM Extension

COBOL has special language features that assist in sort and merge operations so that the user need not program these operations in detail.

Sort/Merge Concepts

Sorting and merging have always constituted a large percentage of the workload in business data processing. COBOL standardizes the specification of these operations, making them easy to specify and modify. In addition, the COBOL user can alternatively use the System/38 logical file support to perform these operations as separate command language (CL) commands. The COBOL language supports these operations through the file-control entry in the Environment Division, the SD

(sort-merge-file-description) entry in the Data Division, and the SORT and MERGE statements in the Procedure Division.

The sort or merge file is described through the file-control entry in the Environment Division, and the SD entry in the Data Division. The sort or merge file is the working file used during the sort or merge; it can be considered an internal file. As such, blocking and internal storage allocation for this file are not under the control of the COBOL user. However, a sort or merge file, like any file, is a set of records, and a sort-merge file description can be considered a particular type of file description.

The sort-merge file is processed through a Procedure Division SORT or MERGE statement. The statement specifies the key field(s) within the record upon which the sort or merge is to be arranged. Keys can be specified as ascending or descending. When more than one key is specified, a mixture of the two sequences is allowed. The sequence of sorted or merged records conforms to the mixture of keys specified.

Sort Concepts

Through the SORT statement, the COBOL user has access to input procedures (used before sorting) and output procedures (used after sorting) that can add, delete, alter, edit, or otherwise modify the records in the input and/or output files. A COBOL program can contain any number of sorts, each with its own independent input and/or output procedures. During SORT statement execution, these procedures are automatically executed at the specified point in processing; thus, extra passes through the sort file are avoided.

A COBOL program containing a sort is usually organized so that one or more input files are read and operated on by an input procedure. Within the input procedure a RELEASE statement (analogous to the WRITE statement) places a record in the sort file. That is, when input procedure execution is completed, a sort file has been created by placing records one at a time into the sort file through the RELEASE statement. If the user does not wish to modify the records before the sorting operation begins, the SORT statement USING phrase releases the unmodified records to the sort file.

After all the input records have been placed in the sort file, the sorting operation is executed. This operation arranges the entire set of sort file records in the sequence specified by the key(s).

After completion of the sorting operation, sorted records can be made available from the sort file, one at a time, through a RETURN statement for modification in an output procedure. If the user does not wish to modify the sorted records, the SORT statement GIVING option names the sorted output file.

Merge Concepts

Through the MERGE statement, the COBOL user has access to output procedures (used after merging) that can modify the records in the output file. The COBOL program can contain any number of merge operations, each with its own independent output procedures. During MERGE statement execution, these procedures are automatically executed at the specified point in processing.

The merge operation compares keys within the records of the input files and arranges the records within the merged file in the sequence specified by the key(s).

Merged records can then be made available, one at a time, through a RETURN statement for modification in an output procedure. If the user does not wish to modify the merged records, the MERGE statement GIVING phrase names the merged output file.

Environment Division – SORT/MERGE

In the Environment Division, the user must write file-control entries for each file used as input to or output from a sort or merge operation. The user must also write a file-control entry for each unique sort-file or merge-file.

File-Control Paragraph

See “FILE-CONTROL Paragraph” in Chapter 3 for a description of input and output files of a sort or merge operation.

I-O-Control Paragraph

In the I-O-Control paragraph, the SAME SORT AREA or SAME SORT-MERGE AREA clause is used.

Format

$$\left[\text{SAME} \left[\begin{array}{l} \text{RECORD} \\ \text{SORT} \\ \text{SORT-MERGE} \end{array} \right] \text{AREA FOR file-name-2 } \{, \text{file-name-3} \} \dots \right] \dots$$

The SAME SORT AREA and SAME SORT-MERGE AREA clauses are syntax-checked, but are treated as documentation.

Restrictions on the specification of SAME RECORD AREA clause are given under “I-O-Control Paragraph” in Chapter 3.

Data Division – SORT/MERGE

In the File Section, the user must write an FD entry for each file that is input to or output from the sort/merge operation, as well as a record description entry. In addition, there must be an SD (sort-merge-file-description) entry for each sort or merge file.

Format

```
[ SD file-name  
  [ RECORD CONTAINS [ integer-1 TO ] integer-2 CHARACTERS ]  
  [ DATA { RECORD IS  
           RECORDS ARE } data-name-1 [ , data-name-2 ] . . . ] .  
  { record-description-entry } . . . ]
```

The level indicator SD identifies the beginning of the SD entry, and must precede the file-name. The file-name must specify a sort or merge file.

The clauses that follow file-name are optional, and their order of appearance is not significant. Both the RECORD CONTAINS clause and the DATA RECORDS clause are described in Chapter 4.

One or more record description entries must follow the SD entry. However, no input/output statements may be executed for this file.

The following example illustrates the File Section entries needed for a sort or merge file:

```
SD SORT-FILE.  
01 SORT-RECORD PICTURE X(80).
```

Procedure Division – SORT/MERGE

The Procedure Division contains MERGE and SORT statements to describe the merge and sort operations and, optionally, sort input procedures and/or sort/merge output procedures. A sort input procedure must contain a RELEASE statement that makes each record available to the sorting operation. A sort/merge output procedure must contain a RETURN statement that makes a sorted/merged record available to the output procedure.

The Procedure Division can contain more than one SORT and/or MERGE statement. These statements can appear anywhere except in the Declaratives portion or in the sort input or sort/merge output procedures.

Files specified in the USING and GIVING phrases of the SORT and MERGE statements must be described explicitly or implicitly in their file-control entries as having sequential organization.

USE procedures are not executed if they reference files specified on a USING or GIVING phrase of a SORT or MERGE statement. If these files are also referenced in an I-O statement within an output procedure or a SORT input procedure, a USE procedure for the file specified is invoked when necessary.

MERGE Statement

The MERGE statement combines two or more identically sequenced files that have already been sorted in an identical ascending/descending key sequence on one or more keys. This statement makes records available in merged order to an output procedure or output file.

Format

```

MERGE file-name-1 ON { ASCENDING
                     DESCENDING } KEY data-name-1 [ , data-name-2 ] . . .
                    [ ON { ASCENDING
                          DESCENDING } KEY data-name-3 [ , data-name-4 ] . . . ] . . .
                    [ COLLATING SEQUENCE IS alphabet-name ]
                    USING file-name-2, file-name-3 [ , file-name-4 ] . . .
                    { OUTPUT PROCEDURE IS section-name-1 [ { THROUGH
                                                            THRU } section-name-2 ] }
                    [ GIVING file-name-5 ]

```

File-name-1 is the name given in the SD entry that describes the records being merged. No file-name may be repeated in the MERGE statement.

When the MERGE statement is executed, all records contained in file-name-2, file-name-3, and so on, are accepted by the sort/merge program and then merged according to the key(s) specified. These files must not be open when the MERGE statement is executed; they are automatically opened and closed by the MERGE operation, and all implicit functions are performed. The files are closed as if the CLOSE statement were written without any optional processing.

See “MERGE Statement and SORT Statement Phrases” later in this chapter for details about the phrases of the MERGE statement.

SORT Statement

The SORT statement accepts records from one or more files, sorts them according to the specified key(s), and makes records available either through an output procedure or in an output file.

Format

```
SORT file-name-1 ON { ASCENDING  
                          DESCENDING } KEY data-name-1 [ , data-name-2 ] . . .  
  
                          [ ON { ASCENDING  
                                  DESCENDING } KEY data-name-3 [ , data-name-4 ] . . . ] . . .  
  
                          [ COLLATING SEQUENCE IS alphabet-name ]  
  
                          { INPUT PROCEDURE IS section-name-1 [ { THROUGH  
                                                                          THRU } section-name-2 ] }  
                          { USING file-name-2 [ , file-name-3 ] . . . }  
  
                          { OUTPUT PROCEDURE IS section-name-3 [ { THROUGH  
                                                                          THRU } section-name-4 ] }  
                          { GIVING file-name-4 }
```

File-name-1 is the name given in the SD entry that describes the records being sorted.

When the SORT statement is executed, all records contained in file-name-2, file-name-3, and so on are accepted by the sort/merge program and then sorted according to the key(s) specified. These input files must not be open at the time the SORT statement is executed; they are automatically opened and closed by the SORT operation, and all implicit functions are performed. The files are closed as if the CLOSE statement were written without any optional processing.

MERGE Statement and SORT Statement Phrases

Most SORT/MERGE statement phrases apply to both the SORT and the MERGE statements. The common SORT/MERGE statement phrases are the ASCENDING/DESCENDING KEY phrase, the COLLATING SEQUENCE phrase, the USING phrase, the GIVING phrase, and the OUTPUT PROCEDURE phrase. The INPUT PROCEDURE phrase applies only to the SORT statements.

ASCENDING/DESCENDING KEY Phrase

This phrase specifies that records are to be processed in an ascending or descending key sequence based on the specified sort/merge keys.

Each data-name specifies a KEY data item on which the sort-merge will be based. Each such data-name must identify a data item in a record associated with file-name-1. The following rules apply:

- A specific KEY data item must be physically located in the same position and have the same data format in each input file; however, it need not have the same data-name.

- If file-name-1 has more than one record description, then the KEY data items need be described in only one of the record descriptions.
- KEY data items must be fixed-length items.
- KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.
- The total length (in bytes) of the KEY data items must not exceed 248.
- KEY data items can be qualified; they cannot be subscripted or indexed.

The KEY data items are listed in order of decreasing significance, regardless of how they are divided into KEY phrases. Using the SORT format as an example, data-name-1 is the most significant key and records are processed in ascending or descending order on that key; data-name-2 is the next most significant key and within data-name-1 records are processed on data-name-2 in ascending or descending order. Within data-name-2, records are processed on data-name-3 in ascending or descending order; within data-name-3, records are processed on data-name-4 in ascending or descending key sequence.

The direction of the sort/merge operation depends on the specification of the ASCENDING or DESCENDING keywords as follows:

- When ASCENDING is specified, the sequence is from the lowest key value to the highest key value.
- When DESCENDING is specified, the sequence is from the highest key value to the lowest.
- If the KEY data item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, the sequence of key values depends on the collating sequence used.
- The key comparisons are performed according to the rules for comparison of operands in a relation condition. See “Simple Conditions, Relation Condition” in Chapter 5.

COLLATING SEQUENCE Phrase

This phrase specifies the collating sequence to be used in nonnumeric comparisons for the KEY data items in this sort/merge operation.

Alphabet-name must be specified in the SPECIAL-NAMES paragraph alphabet-name clause. Any one of the alphabet-name clause phrases can be specified with the following results:

- When NATIVE is specified, the EBCDIC collating sequence is used for all nonnumeric comparisons.
- When STANDARD-1 is specified, the ASCII collating sequence is used for all nonnumeric comparisons.

- When the literal phrase is specified, the collating sequence established by the specification of literals in the alphabet-name clause is used for all nonnumeric comparisons.

When the COLLATING SEQUENCE phrase is omitted, the PROGRAM COLLATING SEQUENCE clause (if specified) in the OBJECT-COMPUTER paragraph specifies the collating sequence to be used. When both the COLLATING SEQUENCE phrase and the PROGRAM COLLATING SEQUENCE clause are omitted, the EBCDIC collating sequence is used.

USING Phrase

When the USING phrase is specified, all input files are transferred automatically to file-name-1. At the time the SORT or MERGE statement is executed, these files must not be open; the COBOL compiler opens, reads, makes records available, and closes these files automatically. If EXCEPTION/ERROR procedures are specified for these files, the COBOL compiler makes the necessary linkage to these procedures.

The input files must have sequential organization.

All input files must be described in an FD entry in the Data Division, and their record descriptions must describe records of the same size as the record described for the sort or merge file. If the elementary items that make up these records are not identical, the user must describe the input records as having the same number of character positions as the sort record.

GIVING Phrase

When the GIVING phrase is specified, all the sorted or merged records in file-name-1 are automatically transferred to the output file (MERGE file-name-5 or SORT file-name-4). At the time the SORT or MERGE statement is executed, this file must not be open; the COBOL compiler opens, writes, and closes the output file automatically. The records overwrite the previous contents, if any, of the file.

If file-name-1 is a logical data base file, the records are added to the end of the file.

End of IBM Extension

If EXCEPTION/ERROR procedures are specified for the output file, the COBOL compiler makes the necessary linkage to these procedures.

The output file must have sequential organization.

The output file must be described in an FD entry in the Data Division, and its record description(s) must describe records of the same size as the record described for the sort or merge file. If the elementary items that make up these records are not identical, the user must describe the output record as having the same number of character positions as the sort or merge record.

SORT INPUT PROCEDURE Phrase

This phrase specifies the section-name(s) of a procedure that is to modify input records before the sorting operation begins.

Section-name-1 specifies the first (or only) section in the input procedure. Section-name-2 (when specified) identifies the last section of the input procedure.

The input procedure must consist of one or more sections that are written consecutively and do not form a part of any output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file.

Control must not be passed to the input procedure except when a related SORT statement is being executed because the RELEASE statement in the input procedure has no meaning unless it is controlled by a SORT statement. The input procedure can include any procedures needed to select, create, or modify records. The following restrictions apply to the procedural statements within an input procedure:

- The input procedure must not contain any SORT or MERGE statements.
- The input procedure must not contain any transfers of control to points outside the input procedure. The execution of a CALL statement to another program, or the execution of USE Declaratives is not considered a transfer of control outside an input procedure. Hence, they are allowed to be activated within these procedures.

IBM Extension

- If control transfers via a PERFORM statement to a point outside the input procedure, a conditional level message is issued but compilation continues.

End of IBM Extension

- The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedure with the exception of the return of control from a Declaratives Section.

IBM Extension

- If control transfers via a PERFORM statement to a point inside the input procedure from elsewhere in the Procedure Division, a conditional level message is issued but the compilation continues.

End of IBM Extension

If an input procedure is specified, control is passed to the input procedure when the SORT program input phase is ready to receive the first record. The compiler inserts a return mechanism at the end of the last section of the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted. The RELEASE statement transfers records from the Input Procedure to the sort file, which is then used in the input phase of the sort operation.

SORT/MERGE OUTPUT PROCEDURE Phrase

This phrase specifies the section-name(s) of a procedure that is to modify output records from the sort or merge operation.

Section-name-3 specifies the first (or only) section in the output procedure. Section-name-4 (when specified) identifies the last section of the output procedure.

The output procedure must consist of one or more sections that are written consecutively and are not part of any input procedure. The output procedure must include at least one RETURN statement in order to make sorted/merged records available for processing.

When all the records are sorted/merged, control is passed to the output procedure. The RETURN statement in the output procedure is a request for the next record.

Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed because RETURN statements in the output procedure have no meaning unless they are controlled by a SORT or MERGE statement. The output procedure can consist of any procedures needed to select, modify, or copy the records that are being returned one at

a time from the sort/merge file. There are three restrictions on the procedural statements within the output procedure:

- The output procedure must not contain any SORT or MERGE statements.
- The output procedure must not contain any transfers of control to points outside the output procedure. The execution of a CALL statement to another program, or the execution of USE Declaratives are not considered as transfers of control outside an output procedure. Hence, they are allowed to be activated within these procedures.

IBM Extension

- If control transfers via a PERFORM statement to a point outside the output procedure, a conditional level message is issued but compilation continues.

End of IBM Extension

- The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedure with the exception of the return of control from a Declaratives Section.

IBM Extension

- If control transfers via a PERFORM statement to a point inside the input procedure from elsewhere in the Procedure Division, a conditional level message is issued but the compilation continues.

End of IBM Extension

When an output procedure is specified, control passes to it after the sort/merge file (file-name-1) has been placed in sequence by the sort/merge operation. The COBOL compiler inserts a return mechanism at the end of the last section in the output procedure; when control is passed to the last statement in the output procedure, the return mechanism terminates the sort or merge, and passes control to the next executable statement after the SORT or MERGE statement.

SORT or MERGE INPUT/OUTPUT PROCEDURE Control

The INPUT or OUTPUT PROCEDURE phrases function in a manner similar to Format 1 of the PERFORM statement (the simple PERFORM). For example, naming a section in an OUTPUT PROCEDURE phrase causes execution of that section during the sort/merge operation to proceed as if that section were named in a PERFORM statement. As with the PERFORM statement, execution of the section is terminated after execution of its last statement. The last statement in Input and Output Procedures can be the EXIT statement. This is useful for documentation purposes.

RELEASE Statement (Sort Function Only)

The RELEASE statement transfers records from an input/output area to the initial phase of a sort operation. This statement is similar to the WRITE statement.

The RELEASE statement can be specified only within an input procedure associated with a SORT statement. Within an input procedure at least one RELEASE statement must be specified.

When the RELEASE statement is executed, the current contents of record-name are placed in the sort file; that is, made available to the initial phase of the sort operation.

Format

`RELEASE record-name [FROM identifier]`

Record-name must specify a record associated with the SD entry for file-name-1. Record-name can be qualified.

When the FROM identifier phrase is specified, the RELEASE statement is equivalent to the statement MOVE identifier to record-name followed by the statement RELEASE record-name. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING phrase.

Identifier and record-name must not refer to the same storage area.

After the RELEASE statement is executed, the information in record-name is no longer available unless file-name-1 is specified in a SAME RECORD AREA clause, in which case record-name is still available as a record of the other files named in that clause. When the FROM identifier phrase is specified, the information is still available in identifier.

When control passes from the input procedure, the sort file consists of all those records placed in it by execution of RELEASE statements.

RETURN Statement

The RETURN statement transfers records from the final phase of a sort or merge operation to an input/output area. This statement is similar to the READ statement.

The RETURN statement can be specified only within an output procedure associated with a SORT or MERGE statement. Within an output procedure at least one RETURN statement must be specified.

Format

RETURN file-name RECORD [INTO identifier] AT END imperative-statement

When the RETURN statement is executed, the next record from file-name is made available for processing by the output procedure.

File-name must be described in a Data Division SD entry.

If more than one record description is associated with file-name, these records automatically share the same storage; that is, the area is implicitly redefined. After RETURN statement execution, only the contents of the current record are available; if any data items lie beyond the length of the current record, their contents are undefined.

When the INTO identifier phrase is specified, the RETURN statement is equivalent to the statement RETURN file-name followed by the statement MOVE record-name TO identifier. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to identifier.

The record areas associated with file-name and identifier must not be the same storage area.

After all records have been returned from file-name, the AT END imperative-statement is executed, and no more RETURN statements can be executed.

SORT/MERGE Programming Notes

Sort/merge execution time can vary greatly and depends on the following factors:

- The size of the storage pool in which the job executes
- The number of records sorted
- Record size
- The number of key fields specified
- The collating sequence used
- The number and types of input files
- Message severity.

Storage Pool Size: A minimum storage pool size of 200 K is recommended, and 300 K is recommended for the average storage pool size. Executing sort/merge operations and other applications, especially interactive applications, in the same storage pool increases response time and increases sort/merge execution time.

Number of Records: Execution time increases when the number of records included in the sort increases. The sort/merge operation builds a work record for each input record in the sort. For efficiency, pass only required records to the sort.

Record Size: Execution time increases when the records are longer. To minimize execution time, do not include fields that contain unnecessary information.

Number of Key Fields: Execution time increases when there are more sort/merge key fields. As keys, character fields are more efficient than packed or binary signed fields.

Alternate Collating Sequences: Using an alternate collating sequence increases the execution time of the sort. Additional logic is required to change each key field from the standard collating sequence to the alternate collating sequence.

Number and Types of Files: Because the sort must move each record two times (from the file to the work record area, and from the work file to the output file), consider the characteristics of the file.

A logical file on the System/38 can be a subset of a physical file, a combination of physical files, and/or a restructuring of a group of fields from one or more physical files. Because of the added flexibility, use of logical files as input to the sort/merge operation can increase the execution time of the sort/merge operation. If possible, use a physical file for input to the sort/merge.

Processing a file from a unit record device, such as a diskette or a card reader, is much slower than processing a file from the data base or from tape. If a file is heavily used during a job or a series of jobs, execution time can be improved by copying the file into the data base.

Message Severity: In cases where an input or output procedure transfers control to points outside the procedure, or where control transfers to inside an input or output procedure from elsewhere in the Procedure Division, conditional level message CBL0492 is issued, but compilation does not fail.

In some cases, however, it may be desirable to have compilation fail. If desired, you can force such transfers of control to cause the compilation to fail by changing the conditional level message to a severe level message. This can be done by changing the severity of the message to 30 using the Change Message Description (CHGMSGD) CL Command. See the *CL Reference Manual* for more information about the CHGMSGD command. See the *System/38 Messages Guide: COBOL* for more information about message severities.

SOURCE PROGRAM LIBRARY

Prewritten source program entries can be included in a source program at compile time. Thus, an installation can use standard file descriptions, record descriptions, or procedures without recoding them. These entries and procedures can be saved in files. They can be included in the source program by means of the COPY statement.

COPY Statement

The COPY statement places previously written text in a COBOL program.

Format

$$\text{COPY text-name } \left[\begin{array}{c} \{ \text{OF} \\ \text{IN} \} \end{array} \right] \text{ System/38 file name } \left[- \text{System/38 library name} \right] \\ \left[\text{REPLACING } \left\{ \begin{array}{l} \text{==pseudo-text-1==} \\ \text{identifier-1} \\ \text{literal-1} \\ \text{word-1} \end{array} \right\} \text{ BY } \left\{ \begin{array}{l} \text{==pseudo-text-2==} \\ \text{identifier-2} \\ \text{literal-2} \\ \text{word-2} \end{array} \right\} \dots \right]$$

Compilation of the source program containing COPY statements is logically equivalent to processing all COPY statements before processing the resulting source program.

The effect of processing a COPY statement is that the text associated with text-name is copied into the source program, logically replacing the entire COPY statement beginning with the word COPY and ending with the period, inclusive. When the REPLACING phrase is not specified, the text is copied unchanged.

The text-name is the name of the member to be copied. The text-name must begin with an alphabetic character. The first 10 characters of the text-name are used as the member name; these first 10 characters must, therefore, be unique within one file.

If text-name is not qualified, QCBLSRC is assumed as the System/38 file name. If the System/38 file name is not qualified by System/38 library name, it is assumed to reside in a library in the library list, *LIBL.

The System/38 library name, System/38 file name, and text-name must follow the rules for formation of any System/38 name.

A COPY statement can appear in the source program anywhere that a character-string or a separator can appear. However, a COPY statement must not be specified within the resulting copied text. Each COPY statement must be preceded by a space, and followed by a period and a space.

Comment lines can appear in copied text. Comment lines in text are copied into the source program unchanged and are interpreted logically as a single space.

Debugging lines can appear in copied text. When a COPY statement is specified on a debugging line, the copied text is treated as though it appeared on a debugging line except that comment lines in the text appear as comment lines in the resulting source program.

The syntactic correctness of the entire COBOL source program cannot be determined until all COPY statements have been completely processed, because the syntactic correctness of the copied text cannot be independently determined.

Text copied from a member is placed into the same area of the resultant program as it is in the member. Copied text must conform to the rules for standard COBOL format.

IBM Extension

The COPY statement, DDS or DD format, can also be used to create equivalent COBOL Data Division statements for a file that exists on the system. These descriptions are based on the file in existence at compile time. This statement does not use the DDS source statements for the file. The DDS or DD format of the COPY statement can be used only in the Data Division, and it is the user's responsibility to precede the statement with a group level item that has a level-number less than 05.

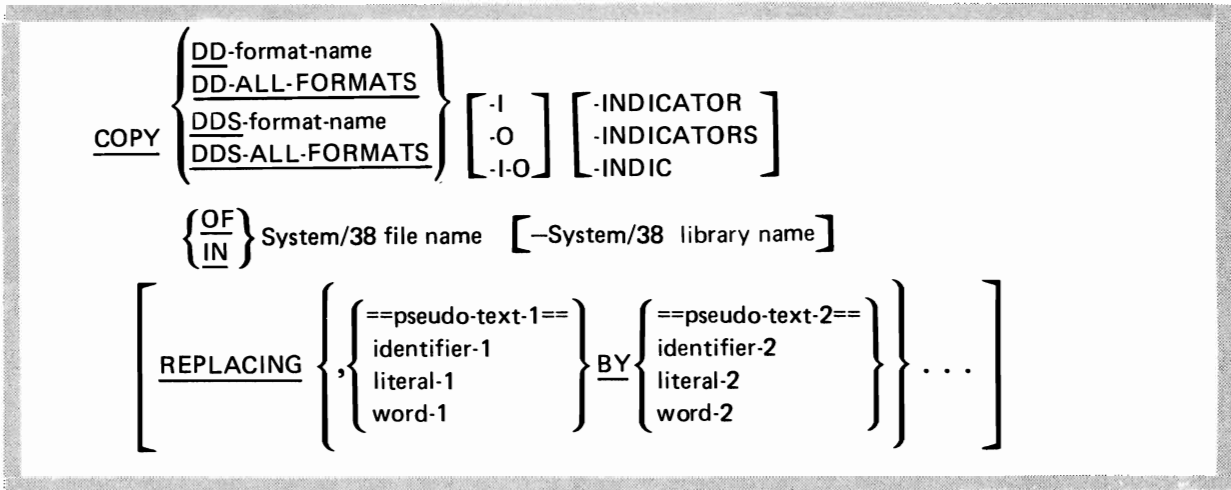
The COPY statement, DD format, can be used to reference an Alias (alternate) name. The specification of an Alias name in DDS allows a data name of up to thirty characters to be included in the COBOL program.

When the COPY statement DD format is specified, COBOL copies in Alias names and translates all underscores to hyphens in the alias names before any replacing occurs.

Note: Alias names which are included in the COBOL source file, can be used in the Procedure Division.

When the RECORD KEY clause specifies EXTERNALLY-DESCRIBED-KEY, a format can be copied only once under an FD. For example, if all formats are copied under an FD, no other COPY statement, DDS or DD format, can be specified for the same file under that FD.

Format 2



The format-name is the name of the DDS record format definition that is translated into COBOL equivalent data description entries. The format-name must follow the rules for formation of any System/38 name.

If DDS-ALL-FORMATS or DD-ALL-FORMATS is specified or -I-O is specified or assumed, each record format is generated as a redefinition of an 05 elementary item defined as the size of the largest record format definition.

If a separate storage area is needed in WORKING-STORAGE for each format, individual COPY statements must be specified for each format.

For example:

```

SELECT FILE-X
ASSIGN TO DATABASE-CUSTMASTER.
.
.
.
FD FILE-X
   LABEL RECORDS ARE STANDARD.
01 FILE-X-RECS.
   COPY DDS-ALL-FORMATS OF
     CUSTMASTER-QGPL. (See Note 1.)
.
.
.
WORKING-STORAGE SECTION.
01 ADR-REC.
   COPY DDS-CUSTADR OF
     CUSTMASTER. (See Note 2.)
01 DETAIL-REC.
   COPY DDS-CUSTDETL OF
     CUSTMASTER. (See Note 2.)

```

Notes:

1. *This COPY statement generates only one storage area for all formats.*
2. *This COPY statement generates separate storage areas.*

System/38 file name is the name of a System/38 file. The generated DDS entries represent the record format defined in the file. The file must be created before the program is compiled.

System/38 library name is optional. If it is not specified, the current job library list is used as the default value.

If neither -I nor -O is specified, -I-O is assumed.

If -I is specified, the generated data description entries contain either the input and input/output fields for a nonsubfile format, or the input, output, and input/output fields for a subfile format.

If -O is specified, the generated data description entries contain one of the following:

- The output and input/output fields for the nonsubfile format.
- The input, output, and input/output fields for the subfile format.

The use of the INDICATOR attribute is discussed under "Indicators" in Chapter 7.

Data base files never have indicators.

Group level names are assigned as follows:

- INPUT
05 format-name-I
- OUTPUT
05 format-name-O

If indicators are requested, or exist in the format, an additional group name (06 level) is generated at the beginning of the structure.

```
06 format-name-(I or O)-INDEC.  
07 IN01 PIC 1 INDIC 01.  
07 IN04 PIC 1 INDIC 04.  
07 IN05 PIC 1 INDIC 05.  
07 IN07 PIC 1 INDIC 07.  
06 FLD1 PIC ...  
06 FLD2 PIC ...
```

If redefinition is required, a group level name is generated as follows:

```
05 file-name-RECORD  
   PIC X(largest record size).
```

Field names, PICTURE definitions, and numeric usage clauses are derived directly from the internal DDS format field names and data type representations. Field names and PICTURE definitions are constructed as follows:

Data Field Structures

```
06 field-name PIC (appropriate COBOL definition...see Figure 6-5).
```

End of IBM Extension

Floating Point Fields

COBOL treats floating point fields as FILLER. The fields can contain floating point values set outside of COBOL, and a COMP-4 definition is generated to maintain proper alignment in the record, but the data is *not* in binary format. No attempt must be made to use floating point data for processing in the COBOL program.

Programming Note: If you have not specified your own program collating sequence, you may create a record containing floating point fields in your COBOL program by moving LOW-VALUES to the entire record before moving in the values of the non-floating-point fields. This will give the floating point fields in the record a value of zero. Note that the above method is only recommended if valid floating point fields with a value of zero are desirable for your particular application.

Floating point key fields are not allowed. In cases where some formats exist without a floating point key field and other formats do not, you should use one or more COPY DD or COPY-DDS statements with specific format names, rather than using COPY DDS-ALL-FORMATS or COPY DD-ALL-FORMATS.

Indicator Structures

07 INxx PIC 1 INDIC xx.

where xx is indicator number.

PHYSICAL, LOGICAL, PRINTER, COMMUNICATIONS, AND BSC FILES			
DDS		COBOL DATA DIVISION n = total field length (DDS pos. 30-34) m = number of decimals (DDS pos. 36 & 37)	
Data Type (Pos. 35)	Formats	If DDS pos. 36 & 37 are blank	If DDS pos. 36 & 37 are not blank
		␣ (Blank) P S B F A	Default Packed decimal Zoned decimal/signed numeric Binary Floating Point ¹ — single precision — double precision Character
DISPLAY FILES			
X N Y I W A	Alphabetic Only Numeric Shift Numeric Only Inhibit Keyboard entry Katakana Alphanumeric Shift	PIC X(n) PIC X(n) PIC S9(n) PIC X(n) PIC X(n) PIC X(n)	— PIC S9(n-m)V9(m) PIC S9(n-m)V9(m) PIC S9(n-m)V9(m) — —

¹ COBOL treats floating point fields as FILLER. See *Floating Point Fields*.

Figure 6-5. Data Field Structures

Externally Described COPY Statement, DDS Format, and DD Format Considerations

When all field descriptions are identical, and the user has requested INPUT and OUTPUT fields implicitly or explicitly, only one set of field descriptions is generated. This type of description is annotated with a comment-line reading "I-O FORMAT: format-name" and neither -I nor -O is appended to the record format name. This is always the case for data base and printer device files. For example:

```

01 RCUSREC.
   COPY DDS-CUSREC OF CUSFILE.
*   I-O FORMAT:CUSREC FROM FILE CUSFILE OF LIBRARY CUSLIB      CUSREC
*   THE KEY DEFINITIONS FOR RECORD FORMAT CUSREC
*   NUMBER NAME RETRIEVAL TYPE ALTSEQ
*   0001 ARBAL ASCENDING SIGNED NO
*   0002 AREACD DESCENDING ABSVAL NO
   05   CUSREC.
   06   ARBAL          PIC S9(7)V9(2)      COMP-3.      CUSREC
   06   AREACD        PIC S9(3)           COMP-3.      CUSREC
   06   BOSTAZ        PIC X(1)           CUSREC
   06   CNTCT         PIC X(15)          CUSREC
   06   CRCHKZ        PIC S9(2)          CUSREC
   06   CSTAT         PIC X(1)           CUSREC
   06   CUSTNZ        PIC S9(6)          CUSREC
   06   DLORD         PIC S9(6)          CUSREC
   06   DSCPCZ        PIC S9(2)V9(3)     COMP-3.      CUSREC
   06   INDUS         PIC S9(2)          CUSREC
   06   NAME1         PIC X(25)          CUSREC
   06   NAME2         PIC X(25)          CUSREC
   06   NAME3         PIC X(25)          CUSREC
   06   NAME4         PIC X(25)          CUSREC
   06   PHONE        PIC S9(7)          COMP-3.      CUSREC
   06   PRICIZ        PIC S9(2)          CUSREC
   06   SHPINZ        PIC X(25)          CUSREC
   06   SLSMAZ        PIC X(3)           CUSREC
   06   TAXCDZ        PIC S9(2)          CUSREC
   06   TERMSZ        PIC S9(2)          CUSREC

```

The user should pay particular attention to the REDEFINES clause that may be generated for the ALL-FORMATS or -I-O phrases. Since all formats are redefined on the same area (generally a buffer area), several field names can describe the same area of storage, and unpredictable results can occur if the entire format area is not reinitialized prior to each output operation. Data items that are subordinate to the data item specified in a MOVE CORRESPONDING statement do not correspond and are not moved when they contain a REDEFINES clause or are subordinate to a redefining item.

To avoid reinitialization, multiple COPY statements, DDS or DD format, using -I and -O suffixes can be used to create separate areas of storage in the Working-Storage section for each format or format type (input or output). READ INTO and WRITE FROM statements can be used with these record formats. For example:

```
FD ORDER-ENTRY-SCREEN
01 ORDER-ENTRY-RECORD
```

```
WORKING-STORAGE SECTION.
01 ORDSFL-I-FORMAT
   COPY DDS-ORDSFL-I OF DOESCR.
01 ORDSFL-O-FORMAT
   COPY DDS-ORDSFL-O OF DOESCR.
```

```
PROCEDURE DIVISION.
```

```
READ SUBFILE ORDER-ENTRY-SCREEN NEXT MODIFIED RECORD
   INTO ORDSFL-I-FORMAT FORMAT IS "ORDSFL"
   AT END SET NO-MODIFIED-SUBFILE-RCO TO TRUE.
```

```
MOVE CORR ORDSFL-I TO ORDSFL-O.
REWRITE SUBFILE ORDER-ENTRY-RECORD FROM ORDSFL-O-FORMAT
   FORMAT IS "ORDSFL"
```

If a user-defined field-name is a COBOL reserved word, the suffix -DDS is appended to the field-name. The format-name can be a COBOL reserved word only if the REPLACING phrase is used to change the copied occurrence of the format-name.

The REPLACING phrase of the COPY statement can be used to replace any of the generated COBOL source, including the level-numbers and the format-name. For example:

```
      COPY DDS-MOVE OF DDSFILE-CUSLIB
      REPLICING MOVE BY CUSREC.
*I-O FORMAT:MOVE FROM FILE DDSFILE OF LIBRARY CUSLIB      MOVE
*                                                         MOVE
* THE KEY DEFINITIONS FOR THE RECORD FORMAT MOVE
* NUMBER NAME RETRIEVAL TYPE ALTSEQ
* 0001 CUST ASCENDING DIGIT NO
*   05 CUSREC.
*   06 CUST          PIC X(5)          MOVE
*       CUSTOMER NUMBER              MOVE
*   06 NAME          PIC X(20).       MOVE
*       CUSTOMER NAME                 MOVE
*   06 ADDR          PIC X(20).       MOVE
*       CUSTOMER ADDRESS              MOVE
*   06 CITY          PIC X(20).       MOVE
*       CUSTOMER CITY                 MOVE
*   06 STATE         PIC X(2).        MOVE
*       STATE ABBREVIATION            MOVE
```

When RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified, the REPLACING phrase cannot change the name of a field that is a key.

DDS field names can contain characters that are not allowed in the COBOL language. For example, the field name CUSTN# can be used as an abbreviation for customer number. This format can be used in a COPY statement, DDS or DD format, if the REPLACING phrase is used to change the invalid characters to valid COBOL characters.

```

COPY DDS-ALL-FORMATS of PHEADR2-CUSTLIB
REPLACING ==CUSTN#== BY CUSTNO
I-O FORMAT:HEDR2 FROM FILE PHEADR2 OF LIBRARY CUSLIB
* THE KEY DEFINITIONS FOR RECORD FORMAT HEADR2
* NUMBER NAME RETRIEVAL TYPE ALTSEQ
* 0001 CUSTNO ASCENDING N NO
* 0002 RECCOD DESCENDING ZONE YES
  05 PHEADR2-RECORD PIC X(113).
  05 HEADR2 REDEFINES PHEADR2-RECORD.
    06 CUSTNO PIC S9(6).
    06 ORDERN PIC S9(6).
    06 RECCOD PIC S9(1).
    06 SHIP1 PIC X(25).
    06 SHIP2 PIC X(25).
    06 SHIP3 PIC X(25).

```

|_____ End of IBM Extension _____|

REPLACING Phrase

In the REPLACING phrase, each operand can consist of one of the following: pseudo-text, an identifier, a literal, or a COBOL word. When the REPLACING phrase is specified, each operand-1 from the copied text is replaced by its associated operand-2.

Pseudo-text is a sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters (= =). Both characters of each pseudo-text delimiter must appear on one line; however, character-strings within pseudo-text can be continued.

Pseudo-text-1 must not be null; neither can it consist solely of the space character and/or of comment lines.

Pseudo-text-2 can be null. It can consist solely of space characters and/or comment lines.

Each identifier can be defined in any Data Division section.

Each literal can be numeric or nonnumeric.

Each COBOL word can be any single COBOL word.

Programming Note: Sequences of code (such as file and data descriptions, error and exception routines, and so on) that are common to a number of programs can be cataloged and used in conjunction with the COPY statement. If naming conventions are established for such common code, then the REPLACING phrase need not be specified. If the names will change from one program to another, then the REPLACING phrase can be used to supply meaningful names for this program.

REPLACING Phrase Processing

When the REPLACING phrase is specified, the text is copied, and each properly matched occurrence of operand-1 within the text is replaced by the associated operand-2.

For purposes of matching, each identifier-1, literal-1, or word-1 is treated as pseudo-text containing only identifier-1, literal-1, or word-1 respectively. Separator spaces in identifiers are optional in both the copied text and the comparison text.

The comparison proceeds as follows:

- Any separator comma, semicolon, and/or space preceding the leftmost word in the text is copied into the source program. Beginning with the leftmost text word and the first operand-1 specified in the REPLACING phrase, the entire REPLACING operand that precedes the keyword BY is compared to an equivalent number of contiguous text words.
- Operand-1 matches the text only if the ordered sequence of text words in operand-1 is equal, character for character, to the ordered sequence of words. For matching purposes, each occurrence of a comma or semicolon separator is considered to be a single space. However, when operand-1 consists solely of a separator comma or semicolon, it participates in the match as a text word. In this case, the space following the comma or semicolon separator can be omitted. Each sequence of one or more space separators is considered to be a single space.
- If no match occurs, the comparison is repeated with each successive operand-1 (if specified) until either a match is found or there are no further REPLACING operands.
- Whenever a match occurs between operand-1 and the copied text, the associated operand-2 is copied into the source program in the place of operand-1.

IBM Extension

Operand-2 is copied in the place of operand-1 unless pseudo-text-2 positioning rules cause the replacement to be inserted in a different area.

End of IBM Extension

- When all operands have been compared and no match is found, the leftmost text word is copied into the source program.
- The next successive uncopied text word is then considered the leftmost text word, and the comparison process is repeated, beginning with the first operand-1. The process continues until the rightmost copied text word has been compared.

- A comment line occurring in operand-1 and in the copied text is interpreted for matching purposes as a single space. A comment line appearing in operand-2 is copied unchanged into the source program.
- Debugging lines are not permitted in operand-1. Debugging lines, however, are permitted in copied text and in operand-2. Text words in a debugging line are matched as if no D appeared in column 7.
- Text words after replacement are placed in the source program according to standard COBOL format rules.

Notes:

1. *Arithmetic and logical operators are considered to be text words and can be replaced only through the pseudo-text phrase.*
2. *When a figurative constant is operand-1, it will match only exactly as specified. For example, if ALL "AB" is specified in the copied text, then "ABAB" is not considered a match. Only ALL "AB" is considered a match.*

COPY Statement Example

In this example, the member PAYREC consists of the following Data Division entries:

```
02 B PIC S99.
02 C PIC S9(5)V99.
02 D PIC S9999 OCCURS 1 TO 52 TIMES
    DEPENDING ON B OF A.
```

The user can use the COPY statement in the Data Division of a program as follows:

```
01 PAYROLL. COPY PAYREC OF PAYFILE.
```

In this program, the member is then copied. The resulting entry is treated as if it had been written as follows:

```
01 PAYROLL.
02 B PIC S99.
02 C PIC S9(5)V99.
02 D PIC S9999 OCCURS 1 TO 52 TIMES
    DEPENDING ON B OF A.
```

To change some (or all) of the names within the member, the user can use the REPLACING phrase:

```
01 PAYROLL. COPY PAYREC OF PAYFILE
    REPLACING A BY PAYROLL
             B BY PAY-CODE
             C BY GROSS-PAY.
```

In this program, the member is copied. The resulting entry is treated as if it had been written as follows:

```
01 PAYROLL.
02 PAY-CODE    PIC S99.
02 GROSS-PAY  PIC-S9(5)V99.
```

02 D

PIC S9999 OCCURS 1 TO 52
TIMES DEPENDING ON
PAY-CODE OF PAYROLL.

The changes shown are made only for this program. The entry as it appears in the member remains unchanged.

SEGMENTATION FEATURE

It is not necessary to be concerned with storage management when writing System/38 COBOL programs. Segmentation, however, is available for compatibility with other systems.

The segmentation feature provides programmer-controlled storage optimization of the Procedure Division by allowing that division to be subdivided both physically and logically.

Segmentation Concepts

Although it is not required, the Procedure Division of a source program is usually written as a consecutive group of sections, each of which is made up of a series of related operations that perform a particular function. Thus, the entire Procedure Division is made up of a number of logical subdivisions. Segmentation allows the programmer to physically divide the Procedure Division into segments, each of which has specific physical and logical attributes.

When Segmentation is used, the entire Procedure Division must be divided into sections. Each section must then be classified as to its physical and logical attributes. Classification is specified by means of segment-numbers. All sections given the same segment-number make up one program segment.

Segment-numbers must be integers from 0 through 99.

Program Segments

There are three types of program segments: fixed permanent, fixed overlayable, and independent.

Fixed Segments

Fixed permanent segments and fixed overlayable segments make up the fixed portion, the part of the Procedure Division that is logically treated as if it were always physically present in main storage. Fixed-portion segment-numbers must be integers from 0 through 49.

A fixed permanent segment is always made available in its last-used state.

A fixed overlayable segment is logically always in main storage during program execution; therefore, it is always available in its last-used state.

Any overlay of such a segment is transparent to the user. Thus, a fixed overlayable segment is logically identical with a fixed permanent segment.

Independent Segments

Logically, an independent segment can overlay and be overlaid by other segments during program execution.

An independent segment is made available in its initial state the first time control is passed to it (explicitly or implicitly) during program execution.

An independent segment is made available in its initial state during subsequent transfers of control when:

- The transfer is the result of an implicit transfer of control between consecutive statements that are in different segments (that is, when control drops through into the independent segment from the physically preceding segment).
- The transfer is the result of an implicit transfer from a SORT or MERGE statement in one segment to a SORT input procedure or SORT/MERGE output procedure in an independent segment.
- An explicit transfer of control from a section with a different segment-number takes place (as, for example, during the transfer of control in a PERFORM n TIMES statement).

An independent segment is made available in its last-used state during subsequent transfers of control when:

- With the exception of the two preceding kinds of implied transfers, an implicit transfer from a section with a different priority takes place (as, for example, when control is returned to the independent segment from a Declarative procedure).
- An explicit transfer results from an EXIT PROGRAM statement.

Independent segments must be assigned segment-numbers 50 through 99.

Segmentation Logic

In a segmented program, the sections are classified by a system of segment-numbers according to the following criteria:

- Frequency of Reference – Much used sections, or those that must be available for reference at all times, should usually be within fixed permanent segments. Less frequently used sections should usually be within either fixed overlayable or independent segments, depending on the program logic.
- Frequency of Use – 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.

- Logical Relationships – Sections that frequently communicate with each other should be given identical segment-numbers.

Segmentation Control

Except for specific transfers of control, the logical sequence and the physical sequence of program instructions are the same. The compiler inserts any instructions necessary to initialize a segment. It is not necessary to transfer control to the beginning of a segment, or to the beginning of a section within a segment. Instead, control can be transferred to any paragraph in the Procedure Division.

COBOL Source Program Considerations

The following elements of a COBOL source program implement the Segmentation feature:

- The SEGMENT-LIMIT clause in the OBJECT-COMPUTER paragraph of the Environment Division. This clause allows the programmer to control the specification of fixed permanent and fixed overlayable segments.
- Procedure Division segment-numbers, which group sections into segments. The segment numbering scheme also allows specifications of independent segments, fixed permanent segments, and (in conjunction with the SEGMENT-LIMIT clause) of fixed overlayable segments.

Segmentation – Environment Division

In the OBJECT-COMPUTER paragraph, the SEGMENT-LIMIT clause allows the user to reclassify fixed permanent segments while retaining the properties of fixed portion segments for the reclassified segments.

Format

[, SEGMENT-LIMIT IS segment-number] .

The SEGMENT-LIMIT clause allows the programmer to specify certain permanent segments as capable of being overlaid by independent segments without losing the logical properties of fixed portion segments.

Segment-number must be an integer ranging in value from 1 through 49.

When the SEGMENT-LIMIT clause is specified:

- Fixed permanent segments are those with segment-numbers from 0 up to, but not including, the segment-number specified.
- Fixed overlayable segments are those with segment-numbers from the segment-number specified through 49.

For example, if SEGMENT-LIMIT IS 25 is specified, sections with segment-numbers 0 through 24 are fixed permanent segments, and sections with segment-numbers 25 through 49 are fixed overlayable segments.

When the SEGMENT-LIMIT cause is omitted, all sections with segment-numbers 0 through 49 are fixed permanent segments.

Segmentation – Procedure Division

In the Procedure Division of a segmented program, section classification is specified through segment-numbers in the section headers.

Format

section-name SECTION [segment-number] .

All sections with the same segment-number make up one program segment. Such sections need not be contiguous in the source program.

The segment-number must be an integer from 0 through 99.

Segments with segment-numbers 0 through 49 are in the fixed portion of the program. Declarative sections can be assigned only these segment-numbers.

Segments with segment-numbers from 50 through 99 are independent segments.

If the segment-number is omitted from the section header, the segment-number is assumed to be 0.

Segmentation – Special Considerations

When segmentation is used, there are restrictions on the ALTER, PERFORM, SORT, and MERGE statements.

There are also special considerations for calling and called programs.

ALTER Statement

A GO TO statement in an independent segment must not be referred to by an ALTER statement in a different segment. All other uses of the ALTER statement are valid and are performed, even if the GO TO statement referred to is in a fixed overlayable segment.

PERFORM Statement

A PERFORM statement in the fixed portion can have in its range, in addition to any Declarative procedures whose execution is caused within that range, only one of the following:

- Sections and/or paragraphs in the fixed portion
- Sections and/or paragraphs contained within a single independent segment.

A PERFORM statement in an independent segment can have within its range, in addition to any Declarative procedures whose execution is caused within that range, only one of the following:

- Sections and/or paragraphs in the fixed portion
- Sections and/or paragraphs wholly contained in the same independent segment as the PERFORM statement.

SORT and MERGE Statements

If a SORT or MERGE statement appears in the fixed portion, then any SORT input procedures or SORT/MERGE output procedures must appear completely in one of the following:

- The fixed portion
- A single independent segment.

If a SORT or MERGE statement appears in an independent segment, then any SORT input procedures or SORT/MERGE output procedures must appear completely in one of the following:

- The fixed portion
- The same independent segment as the SORT or MERGE statement.

Calling and Called Programs

The CALL statement can appear anywhere within a segmented program. When a CALL statement appears in an independent segment, that segment is in its last-used state when control is returned to the calling program.

INTER-PROGRAM COMMUNICATION FEATURE

Complex data processing problems are often solved by the use of separately compiled but logically interdependent programs which, at execution time, form logical and physical subdivisions of a single run unit. A run unit is the total program necessary to solve a data processing problem; it includes one or more programs, and can include programs from source programs written in languages other than COBOL. See "System/38 Inter-Program

Communication Considerations” in Chapter 9 for more information on COBOL and non-COBOL program communication.

Inter-Program Communication Concepts

When the solution of a problem is subdivided into more than one program, the constituent programs must be able to communicate with each other through transfers of control and/or through reference to common data.

Transfers of Control

In the Procedure Division, a calling program can transfer control to a called program, and a called program can itself transfer control to yet another called program. However, a called program must not directly or indirectly call its caller. For example, if program A calls program B, program B calls program C, and program C then calls program A, the results are unpredictable.

When control is passed to a called program, execution proceeds in the normal way. When a called program processing is completed, the program can either transfer control back to the calling program, call another program, or end the run unit.

Common Data

Program interaction can require that both programs have access to the same data.

In a calling program, the common data items are described in the same manner as other File and Working-Storage Section items. Storage is allocated for these items in the calling program. In a called program, common data items are described in the Linkage Section. Storage is not allocated to them in the called program. Because a calling program can itself be a called program, common data items can be described in the Linkage Section of the calling program. In this case, storage is not allocated for these items in the calling program itself, but rather in the program that called the calling program. For example, program A calls program B which calls program C. Data items in program A can be described in the Linkage Sections of programs B and C, and the one set of data can be made available to all three programs.

When control is transferred from the calling to the called program, the programmer must furnish a list of the common data items in both programs. The sequence of identifiers in both lists determines the match of identifiers between the calling and called programs. A corresponding pair of identifiers in the list names a single set of data that is available to both programs. While the called program is executing, any reference to one of these identifiers is a reference to the corresponding data of the calling program.

COBOL Language Considerations

In the Data Division of the source programs, the programmer defines the common data items to be used by both the calling and called programs. In the calling program, these items can be defined in the File, Working-Storage, or Linkage Sections. In the called program, these items must be defined in the Linkage Section. Common data items need not have the same name and data description, but they must contain the same number of characters.

In the Procedure Division, the list of common data items is established through the USING phrase, which names those data items available to both programs. In the called program, only those items named in the USING list of the called program are available from the data storage of the calling program.

A CALL statement in the calling program transfers control to the first nondeclarative procedural statement in the called program. When the called program has completed execution, control is returned to the calling program by an EXIT PROGRAM statement. The entire run unit can be ended by a STOP RUN statement in either program.

Data Division – Inter-Program Communication

In the Data Division of a called program, the programmer specifies in the Linkage Section those data items that are common with the calling program.

Format

LINKAGE SECTION.

[{⁷⁷₀₁₋₄₉ } data-name/FILLER clause
 [REDEFINES clause]
 [BLANK WHEN ZERO clause]
 [INDICATOR clause]
 [JUSTIFIED clause]
 [OCCURS clause]
 [PICTURE clause]
 [SIGN clause]
 [SYNCHRONIZED clause]
 [USAGE clause] .] ...

 [88 condition-name VALUE clause.] . . .
 [66 RENAMES clause.] . . .

The Linkage Section has meaning only if this program functions under control of a CALL statement that contains the USING phrase, or a call statement from another language.

The Linkage Section describes data available within the calling program and referred to in both the calling and called programs. Items described in the Linkage Section do not have space allocated for them in the called program. Procedure Division references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program. For index-names, no such correspondence is established. Index-name references in the calling and called programs always refer to separate indexes. Index-name values can be passed by first moving them to an index data item and passing that index data item.

Items defined in the Linkage Section can be referred to in the Procedure Division only if they are one of the following:

- Operands of a USING phrase in this program
- Data items subordinate to such a USING phrase operand

- Items associated with such a USING operand (such as condition-names or index-names).

Each Linkage Section record-name and noncontiguous data-name must be unique, because neither can be qualified. Descriptions of each clause valid in the Linkage Section are given under “Data Description” in Chapter 4. The following additional considerations apply.

Record Description Entries

Items that have a hierarchical relationship with one another must be grouped into level-01 records according to the rules for formation of record descriptions. Data description clauses can be used to complete the description of the entry. Except for level-88 condition-names, the VALUE clause must not be specified.

Data Item Description Entries

Items that have no hierarchical relationship with each other can be defined as noncontiguous items with level-number 77. The following clauses are required:

- Level-number 77
- Data-name
- PICTURE or USAGE IS INDEX.

Other data description clauses are optional and, when necessary, can complete the description of the item. Except for level-88 condition-names, the VALUE clause must not be specified.

Procedure Division – Inter-Program Communication

In the Procedure Division, control is transferred between a COBOL program and another System/38 program by means of the CALL statement.

Reference to common data is provided through the USING phrase, which can be specified in the CALL statement and in the Procedure Division header of the called program.

The CANCEL statement releases storage used by a called program.

The EXIT PROGRAM statement allows termination of called program processing. The STOP RUN statement allows termination of the run unit.

CALL Statement

The CALL statement transfers control from one object program to another within the run unit. The calling program must contain a CALL statement at the point where another program is to be called.

Execution of the CALL statement passes control to the first nondeclarative instruction of the called program. Control returns to the calling program at the instruction following the CALL statement.

Called programs themselves can contain CALL statements, but a called program that contains a CALL statement that directly or indirectly calls the calling program gives unpredictable results.

Format

$$\text{CALL } \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left[\text{USING data-name-1} \left[\text{, data-name-2} \right] \dots \right] \\ \left[\text{ON OVERFLOW imperative-statement} \right]$$

Literal-1 must be nonnumeric and must conform to the rules for formation of a program-name. The first 10 characters of the literal are used to make the correspondence between the calling program and the called program. The literal must specify the program-name of the called program.

If literal-1 is specified, the call is classified as a static call because the PROGRAM-ID is determined at compile time.

Identifier-1 must be an alphanumeric data item. Its contents must conform to the rules for formation of a program-name (see "PROGRAM-ID Paragraph" in Chapter 3). The first 10 characters of identifier-1 are used to make the correspondence between the calling and called program.

If identifier-1 is specified, the call is classified as a dynamic call because the PROGRAM-ID is resolved at execution time each time a call is made. If literal-1 is specified, the PROGRAM-ID is resolved only once.

CALL statement execution passes control to the called program which becomes part of the run unit. If a CALL statement names a program that does not exist in the job's library list (*LIBL) at execution-time, an error message is issued.

A called program 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 statement for the called program has been executed.

On all other entries into the called program, it is in its last-used state, and the reinitialization of any items is the responsibility of the user. See Chapter 9 for more information about calling programs.

If there is not enough storage resource available to accommodate the called program in the subsystem, the ON OVERFLOW phrase specifies the action to be taken.

If the ON OVERFLOW phrase is not specified, results are unpredictable.

The user return code is set to 0 at the start of execution of any COBOL program, and before a call is made to another program. See the RTVJOBA and DSPJOB commands in the *CL Reference Manual* for more information about return codes.

USING Phrase

The USING phrase makes data items in a calling program available to a called program. COBOL supports the passing of arguments to other System/38 programs. The attributes of the data passed depends on the definition requirements of the called program.

The following discussion of the USING phrase assumes that the calling and called programs are written in COBOL.

Format

USING data-name-1 [, data-name-2] . . .

In a calling program, the USING phrase is valid for the CALL statement; each USING data-name must be defined as a level-01 or level-77 item anywhere in the Data Division. The maximum number of data-names that can be specified is 30.

IBM Extension

In the USING phrase of the calling program CALL statement, the data-names can have level-numbers other than 01 or 77. These data-names can be indexed or subscripted, and can be qualified.

End of IBM Extension

In a called program entered at the beginning of the nondeclaratives portion, the USING phrase is valid in the Procedure Division header; each USING data-name must be defined as a level-01 or level-77 item in the Linkage Section of the called program.

Formats for these individual items show the correct syntax for specifying the USING phrase.

The USING phrase is specified if, and only if, the called program is to operate under control of a CALL statement and that CALL statement itself contains a USING phrase. That is, for each CALL USING statement in a

calling program, there must be a corresponding USING phrase specified in the called program.

The order of appearance of USING data-names in both calling and called programs determines the correspondence of single sets of data available to both programs. The correspondence is positional and not by name. Corresponding data-names must contain the same number of characters, although their data descriptions need not be the same. For index-names, no correspondence is established; index-names in calling and called programs always refer to separate indexes.

The data-names specified in a CALL USING statement name data items available to the calling program that can be referred to in the called program. A given data-name can appear more than once.

In a called program, USING data-names must be defined in the Linkage Section. Of the items defined in the Linkage Section, only those named in the USING phrase are available to the program. Within the called program, USING data-names are processed according to their definition within this program.

When the USING phrase is specified, the program executes as if each reference to a USING data-name in the called program Procedure Division is replaced by a reference to the corresponding USING data-name in the calling program.

Examples that illustrate the USING phrase are given in Figure 6-6 and in "Inter-Program Communication Feature Examples" later in this chapter.

CPF Graphics Support

System/38 COBOL lets you use the CALL statement to access the following CPF Graphics routines:

- Graphical Data Display Manager (GDDM), a set of graphics primitives for drawing pictures
- Presentation Graphics Routines (PGR), a set of business charting routines.

You access all these graphics routines with the same format of the CALL statement:

Format

```
CALL "GDDM" USING routine-name [ , data-name-1 ] . . .
```

Routine-name is the name of the graphics routine you want to use.

The data-names that follow routine-name are the parameters necessary to use certain graphics routines. The number of parameters that you must

specify varies, depending on which routine you select. When you select a graphics routine, make sure each parameter is the correct size and data type as required by that routine.

The following are examples of calling graphics routines. Remember, you must define each parameter as required by the graphics routine you use.

```
MOVE "FSINIT" TO CPF-GRAPHICS-ROUTINE-NAME.  
CALL "GDDM" USING CPF-GRAPHICS-ROUTINE-NAME.
```

```
MOVE "GSFLD" TO CPF-GRAPHICS-ROUTINE-NAME.  
CALL "GDDM" USING CPF-GRAPHICS-ROUTINE-NAME,  
PIC-ROW, PIC-COL,  
PIC-DEPTH, PIC-WIDTH.
```

For more information about graphics routines and their parameters, see the *CFP Graphics Programmer's Guide* and the *CPF Graphics Reference Manual*.

CANCEL Statement

The CANCEL statement releases the storage occupied by a called program.

Format

$$\underline{\text{CANCEL}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left[\begin{array}{l} \text{, identifier-2} \\ \text{, literal-2} \end{array} \right] \cdot \cdot \cdot$$

Each literal or identifier specified in the CANCEL statement must be nonnumeric. The contents must conform to the rules for formation of a program-name (see "PROGRAM-ID Paragraph" in Chapter 3). The first 10 characters of the literal or identifier are used to make the correspondence between the calling and called program.

Each literal or identifier specified in the CANCEL statement must be the same as the literal or identifier specified in the associated CALL statement(s).

Subsequent to the execution of a CANCEL statement, the program referred to by the statement ceases to have any logical relationship to the program in which the CANCEL statement appears.

The program named in a CANCEL statement can be entered in its initial state again after the CANCEL statement is executed: the program is entered in its initial state if a CALL statement that names the program is executed by any program in the run unit. A CALL statement is the only means by which a logical relationship to a canceled program can be reestablished.

A called program is canceled either by being directly referred to as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.

No action results from the execution of a CANCEL statement that names a program that has not been called in the run unit, or that names a program that was called but is at present canceled. In either case, control passes to the next statement.

If a CANCEL statement names a program that does not exist in the library list, an error message is issued.

Called programs can contain CANCEL statements. However, a called program must not contain a CANCEL statement that directly or indirectly cancels a calling program. In this case control is passed to the next statement.

A program named in a CANCEL statement must not refer to any program that has been called and has not yet returned control to the calling program. A program can, however, cancel a program that it did not call. For example, if A calls B and B calls C, when A receives control, it can cancel C; or if A calls B and A calls C, when C receives control, it can cancel B.

Calling Program Description	Called Program Description
WORKING-STORAGE SECTION. 01 PARAM-LIST. 05 PARTCODE PIC A. 05 PARTNO PIC X(4). 05 U-SALES PIC 9(5). . . PROCEDURE DIVISION. . . CALL CALLED-PROG USING PARAM-LIST.	LINKAGE SECTION. 01 USING-LIST. 10 PART-ID PIC X(5). 10 SALES PIC 9(5). . . PROCEDURE DIVISION USING USING-LIST.

Figure 6-6. Common Data Items in Inter-Program Communication

Note: In the calling program, the code for parts (PARTCODE) and the part number (PARTNO) are referred to separately. In the called program, the code for parts and the part number are combined into one data item (PART-ID); therefore in the called program, a reference to PART-ID is the only valid reference to them.

EXIT PROGRAM Statement

The EXIT PROGRAM statement specifies the logical end of a called program (subprogram).

Format

paragraph-name. EXIT PROGRAM.

The EXIT statement must be preceded by a paragraph-name, and it must be the only statement in the paragraph.

When control reaches an EXIT PROGRAM statement in a subprogram, control returns to the point in the calling program immediately following the CALL

statement. If control reaches an EXIT PROGRAM statement in a main program, control passes through the exit point to the first sentence of the next paragraph. For more information, see "System/38 Inter-Program Communication Considerations" in Chapter 9.

STOP RUN Statement

Format

STOP RUN

When STOP RUN is specified, execution of the run unit is terminated. If a STOP RUN statement appears in a sequence of imperative statements, it must be the last or the only statement in the sequence. All files should be closed before a STOP RUN statement is executed. If you do not close the files, they are closed by compiler generated code. For more information, see "System/38 Inter-Program Communication Considerations" in Chapter 9.

An implicit STOP RUN is always generated after the last statement in the source program.

Inter-Program Communication Feature Examples

A static CALL is illustrated in the following program example.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. CALLSTAT.  
.  
DATA DIVISION.  
.  
WORKING-STORAGE SECTION.  
01 RECORD-2 PIC X.  
01 RECORD-1.  
   05 SALARY PIC S9(5)V99.  
   05 RATE PIC S9V99.  
   05 HOURS PIC S9V99.  
.  
PROCEDURE DIVISION  
.  
CALL "PROG" USING RECORD-1.  
.  
STOP RUN.
```

The following example illustrates a dynamic CALL. The dynamic CALL differs in execution from the static CALL that is illustrated in the preceding example.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. CALLDYNA.
```

```
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.  
01 IDENT PICTURE X(10).
```

```
01 RECORD-2 PIC X.  
01 RECORD-1.  
05 SALARY PIC S9(5)V99.  
05 RATE PIC S9V99.  
05 HOURS PIC XXX.
```

```
PROCEDURE DIVISION.
```

```
MOVE "PROG" TO IDENT.  
CALL IDENT USING RECORD-1.
```

```
CANCEL IDENT.
```

```
STOP RUN.
```

The following called program can be associated with either of the calling programs in the two preceding examples.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. PROG.
```

```
DATA DIVISION.
```

```
LINKAGE SECTION.  
01 PAYREC.  
10 PAY PIC S9(5)V99.  
10 HOURLY-RATE PIC S9V99.  
10 HOURS PIC S99V9.  
01 CODE PIC 9.
```

```
PROCEDURE DIVISION USING PAYREC.
```

```
EXIT PROGRAM.
```

Processing in these examples begins in the calling program, which can be either CALLSTAT or CALLDYNA. When the first CALL statement is executed, control is transferred to the first statement of the Procedure Division in the called program, PROG.

Note that in each of the calling programs the operand of the first USING phrase is identified as RECORD-1.

When PROG receives control, the values within RECORD-1 are made available to PROG; however, in PROG they are referred to as PAYREC.

Note that the PICTURE character-strings within PAYREC and CODE contain the same number of characters as RECORD-1 and RECORD-2, although the descriptions are not identical.

When processing within PROG reaches the EXIT PROGRAM statement, control is returned to the calling program, and processing continues in that program.

In any given execution of these two calling programs, if the values within RECORD-1 are changed between the time of the first CALL and another CALL, the values passed at the time of the second CALL statement are the changed, not the original, values. If the user wishes to use the original values, then he must ensure that they have been saved.

DEBUGGING FEATURES

The debugging features specify the conditions under which procedures are to be monitored during program execution.

COBOL source language debugging statements are provided. The user decides what to monitor and what information to retrieve for debugging purposes. The COBOL debugging features simply provide access to pertinent information.

COBOL Source Language Debugging

COBOL language elements that implement the Debugging Feature are a compile-time switch (WITH DEBUGGING MODE), an execution-time switch, a USE FOR DEBUGGING Declarative, the special register DEBUG-ITEM, and debugging lines that can be written in the Environment, Data, and Procedure Divisions.

Compile-Time Switch

In the SOURCE-COMPUTER paragraph of the Configuration Section, the WITH DEBUGGING MODE clause acts as a compile-time switch.

Format

SOURCE-COMPUTER, computer-name

[WITH DEBUGGING MODE] .

The WITH DEBUGGING MODE clause serves as a compile-time switch for the debugging statements written in the source program.

When WITH DEBUGGING MODE is specified, all debugging sections and debugging lines are compiled as specified in this chapter. When WITH DEBUGGING MODE is omitted, all debugging sections and debugging lines are treated as documentation.

Execution-Time Switch

The execution-time switch dynamically activates the debugging code that is generated when WITH DEBUGGING MODE is specified.

Two commands are provided to control the execution-time switch. To set the execution-time switch on, enter the command:

```
ENTCBLDBG PGM program-name .LIBL  
Library-name
```

To set the execution-time switch off, enter the command:

```
ENDCBLDBG PGM program-name .LIBL  
Library-name
```

The default for the execution-time switch is off.

When debugging mode is specified, through the execution-time switch, all the debugging sections and debugging lines (D in column 7) compiled into the program are activated.

The ENTCBLDBG command must be entered for each COBOL program (main program or called program) to be debugged in the next COBOL run unit. At the end of the run unit, all execution-time switches that are on are set off. If a switch must be set off before a following execution of a COBOL run unit, the ENDCBLDBG command should be used. Execution-time switches for up to 15 programs can be on at once.

When the ENTCBLDBG or ENDCBLDBG command is issued in a CL program, concatenation expressions can be used for all parameter values. See the *CL Reference Manual* for more information about concatenation expressions:

When debugging mode is suppressed, through the execution-time switch, any USE FOR DEBUGGING Declarative procedures are inhibited. However, all debugging lines (D in column 7) remain in effect.

Recompilation of the source program is not required to activate or deactivate the execution-time switch.

When WITH DEBUGGING MODE is not specified in the SOURCE-COMPUTER paragraph, the execution-time switch has no effect on execution of the program.

USE FOR DEBUGGING Declarative

The USE FOR DEBUGGING sentence in the Procedure Division identifies the items in the source program that are to be monitored by the associated debugging Declarative procedure.

Format

USE FOR DEBUGGING ON { [ALL REFERENCES OF] identifier-1
file-name-1
procedure-name-1
ALL PROCEDURES }

[[ALL REFERENCES OF] identifier-2
file-name-2
procedure-name-2
ALL PROCEDURES] . . .

When specified, all debugging sections must be written immediately after the DECLARATIVES header. Except for the USE FOR DEBUGGING sentence there must be no reference to any nondeclarative procedure within the debugging procedure.

Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

A debugging section for a specific operand is executed only once as the result of the execution of a single statement, no matter how many times the operand is specified in the statement. An exception to this rule is that each specification of a subscripted or indexed identifier where the subscripts or indexes are different causes invocation of the debugging Declarative. For a PERFORM statement that causes repeated execution of a procedure, any associated procedure-name debugging Declarative section is executed once for each execution of the procedure.

For debugging purposes, each separate occurrence of an imperative verb within an imperative statement begins a separate statement.

Statements appearing outside the debugging sections must not refer to procedure-names defined within the debugging sections.

Except for the USE FOR DEBUGGING sentence itself, statements within a debugging Declarative section can only refer to procedure-names defined in a different USE procedure through the PERFORM statement. Procedure-names within debugging Declarative sections must not appear in USE FOR DEBUGGING sentences.

Figure 6-7 defines the points during program execution when the USE FOR DEBUGGING procedures are executed. Identifier-n, file-name-n, and procedure-name-n refer to the first and all subsequent specifications of that type of operand in one USE FOR DEBUGGING sentence. Any particular identifier, file-name, or procedure-name can appear in only one USE FOR DEBUGGING sentence, and only once in that sentence.

An identifier in a USE FOR DEBUGGING sentence:

- Must be specified without the subscripting or indexing normally required if it contains an OCCURS clause or is subordinate to an entry containing an OCCURS clause. (A SEARCH or SEARCH ALL statement that refers to such an identifier does not invoke the USE FOR DEBUGGING procedures.)
- Must not be a special register.

When ALL PROCEDURES is specified in a USE FOR DEBUGGING sentence, procedure-name-1, procedure-name-2, procedure-name-3, and so on, must not be specified in any USE FOR DEBUGGING sentence. The ALL PROCEDURES phrase can be specified only once in a program.

When a USE FOR DEBUGGING operand is used as a qualifier, such a reference in the program does not activate the debugging procedures.

References to the DEBUG-ITEM special register can be made only from within a debugging Declarative procedure.

USE FOR DEBUGGING Operand	Upon execution of the following, the USE FOR DEBUGGING procedures are executed immediately
identifier-n	<p>Before REWRITE/WRITE identifier-n and after FROM phrase move, if applicable.</p> <p>After each initialization, modification, or evaluation of identifier-n in PERFORM/VARYING/AFTER/UNTIL identifier-n.</p> <p>After any other COBOL statement that explicitly refers to identifier-n and could change its contents. (See note.)</p>

Figure 6-7 (Part 1 of 2). Execution of Debugging Declaratives

USE FOR DEBUGGING Operand	Upon execution of the following, the USE FOR DEBUGGING procedures are executed immediately
ALL REFERENCES OF identifier-n	<p>Before GO TO DEPENDING ON identifier-n, control is transferred, and before any associated debugging section for procedure-name is executed.</p> <p>Before REWRITE/WRITE identifier-n and FROM phrase move, if applicable.</p> <p>After each initialization, modification or evaluation of identifier-n in PERFORM/VARYING/AFTER/UNTIL identifier-n.</p> <p>After any other COBOL statement explicitly referring to identifier-n. (See note.)</p>
file-name-n	<p>After CLOSE/DELETE/OPEN/START file-name-n.</p> <p>After READ file-name-n where AT END/INVALID KEY was not executed.</p>
procedure-name-n	<p>Before each execution of the named procedure.</p> <p>After execution of an ALTER statement referring to the named procedure.</p>
ALL PROCEDURES	<p>Before each execution of every nondebugging procedure.</p> <p>After execution of every ALTER statement (except ALTER statements in Declarative procedures).</p>

Figure 6-7 (Part 2 of 2). Execution of Debugging Declaratives

Note: Operands acted upon but not explicitly named in such statements as ADD, MOVE, or SUBTRACT CORRESPONDING never cause activation of a USE FOR DEBUGGING procedure when such statements are executed. If identifier-n is specified in a phrase that is not executed, the associated debugging section is not executed.

DEBUG-ITEM Special Register

The DEBUG-ITEM special register provides information for a debugging Declarative procedure. DEBUG-ITEM has the following implicit description.

```

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

```

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

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

- **DEBUG-LINE:** The compiler-generated statement number, right justified and padded on the left with zeros. For example, 000112.
- **DEBUG-NAME:** The first 30 characters of the name causing debugging section execution. All qualifiers are separated by the word OF (subscripts or indexes are not entered in DEBUG-NAME).
- **DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3:** If the DEBUG-NAME is subscripted or indexed, the occurrence number of each level is entered in the respective DEBUG-SUB-n. If the item is not subscripted or indexed, these fields remain spaces.
- **DEBUG-CONTENTS:** Data is moved into DEBUG-CONTENTS as shown in Figure 6-8. DEBUG-CONTENTS is the same size as the largest identifier in the program.

Item Causing Debug Section Execution	DEBUG-LINE Contains Number of COBOL Statement Referring to	DEBUG-NAME Contains	DEBUG-CONTENTS Contains
identifier-n	identifier-n	identifier-n	Contents of identifier-n when control passes to debug section.
file-name-n	file-name-n	file-name-n	For READ: contents of record retrieved. Other references: spaces.
procedure-name-n ALTER reference	ALTER statement	procedure-name-n	procedure-name-n in TO PROCEED TO phrase
GO TO procedure-name-n	GO TO statement	procedure-name-n	
procedure-name-n in SORT/MERGE INPUT/OUTPUT PROCEDURE	SORT/MERGE statement	procedure-name-n	"SORT INPUT" "SORT OUTPUT" "MERGE OUTPUT" as applicable
PERFORM statement transfer of control	This PERFORM statement	procedure-name-n	"PERFORM LOOP"
procedure-name-n in a USE procedure	Statement causing USE procedure execution	procedure-name-n	"USE PROCEDURE"

Figure 6-8 (Part 1 of 2). DEBUG-ITEM Subfield Contents

Item Causing Debug Section Execution	DEBUG-LINE Contains Number of COBOL Statement Referring to	DEBUG-NAME Contains	DEBUG-CONTENTS Contains
Implicit transfer from previous sequential procedure	Previous statement executed in previous sequential procedure (see note)	procedure-name-n	"FALL THROUGH"
First execution of first nondeclarative procedure	Line number of first statement in the procedure	First nondeclarative procedure-name	"START PROGRAM"

Figure 6-8 (Part 2 of 2). DEBUG-ITEM Subfield Contents

Note: If this paragraph is preceded by a section header and control is passed through the section header, the statement number refers to the section header.

Debugging Lines

A debugging line is any line in a source program with a D coded in column 7 (the continuation area). If a debugging line contains nothing but spaces in Area A and Area B, it is considered a blank line.

Each debugging line must be written so that a syntactically correct program results whether the debugging lines are compiled into the program or syntax-checked, but are treated as documentation.

Successive debugging lines are permitted. Debugging lines can be continued. However, each continuation line must contain a D in column 7, and character-strings must not be broken across two lines.

Debugging lines can be specified only after the OBJECT-COMPUTER paragraph.

When the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph, all debugging lines are compiled as part of the object program.

When the WITH DEBUGGING MODE clause is omitted, all debugging lines are syntax-checked, but are treated as documentation.

FIPS FLAGGER

The FIPS (Federal Information Processing Standard) Flagger can be specified. Depending on the compiler option specified, it identifies source statements and clauses that do not conform to a specified level of the federal standard. For information on the FIPS Flagger, refer to Chapter 1 and to the *Messages Guide*: *COBOL*.



Chapter 7. TRANSACTION Files (IBM Extension)

IBM Extension

This chapter describes the System/38 COBOL language extensions that support work stations and program-to-program communication.

The TRANSACTION file organization allows a COBOL program to interactively communicate with:

- One or more work station users
- One or more programs on a remote system
- One or more devices on a remote system
- Any combination of the above.

On the System/38 you communicate with a program or device on a remote system by using LU1, BSC, or PEER devices. For a detailed discussion of these devices, see the *CPF Data Communications Programmer's Guide*.

When required, references are made to prior discussions in earlier chapters of this book, in order to avoid repetition. The language extensions in this chapter are presented in the following sequence: Environment Division, Data Division, and Procedure Division.

Externally Described Transaction File

A COBOL TRANSACTION file normally uses an externally described display file, BSC file, communications file, or mixed file that contains file information and a description of the fields in the records. The records in this file are described to the COBOL program by the DDS format of the COPY statement (see "Source Program Library" in Chapter 6).

In addition to the field descriptions (such as field names and attributes), the data description specifications (DDS) for a display device file:

- Specify the line number and position number entries for each field and constant to format the placement of the record on the screen.
- Specify attention functions such as underlining and highlighting fields, reverse image, or a blinking cursor.

- Specify validity checking for data entered at the display work station. Validity checking functions include:
 - Detecting fields where data is required
 - Detecting mandatory fill fields
 - Detecting incorrect data types
 - Detecting data for a specific range
 - Checking data for a valid entry
 - Performing modulus 10 or 11 check digit verification.
- Control screen management functions such as when fields are to be erased, overlaid, or retained when new data is displayed.
- Associate indicators 01 through 99 with command attention keys or command function keys. If a command key is described as a command function key (CF), both the modified data record and the response indicator are returned to the program. If a command key is described as a command attention key (CA), the response indicator is returned to the program, but the data record usually contains default values for input-only fields and the values written to the format for hidden output/input fields. For more information on CF and CA keys, see the *CPF Programmer's Guide*.
- Assign an edit code (EDTCDE keyword) or edit word (EDTWRD keyword) to a field to specify how the field's values are to be displayed.
- Specify subfiles.

A display device record format contains three types of fields:

- *Input Fields:* Input fields pass from the device to the program when the program reads a record. Input fields can be initialized with a default value; if the default value is not changed, the default value passes to the program. Uninitialized input fields are displayed as blanks into which the work station user can enter data.
- *Output Fields:* Output fields pass from the program to the device when the program writes a record to a display. The program or the record format in the device file can provide output fields.
- *Output/Input (both) Fields:* An output/input field is an output field that can be changed and, therefore, become an input field. Output/input fields pass from the program when the program writes a record to a display and pass to the program when the program reads a record from the display. Output/input fields are used when the user is to change or update the data that is written to the display from the program.

For a detailed description of an externally described display device file or data communications file and for a list of the valid data description specifications (DDS) keywords, see the *CPF Reference Manual – DDS*.

Figure 7-1 shows an example of the DDS for a display device file.

File	Keying Instruction	Graphic							
Programmer	Date	Key							

Description	Page	of
-------------	------	----

Sequence Number	Form Type	And/Or Comment (A/O/')	Conditioning				Name	Length	Reference (R)	Data Type (M A/P/S/B A/S/K/Y/N/I/W)	Decimal Positions	Usage (M/O//B/H/M)	Location		Functions
			Indicator	Not (N)	Indicator	Not (N)							Line	Pos	
1	A	X													
2	A	X													
3	A														
4	A														
5	A														
6	A														
7	A														
8	A														
9	A														
10	A														
11	A														
12	A														
13	A														
14	A														
15	A														
16	A														
17	A														
18	A														
19	A														
20	A														
21	A														
22	A														
23	A														
24	A														
25	A														
26	A														
27	A														
28	A														
29	A														
30	A														
31	A														
32	A														
33	A														
34	A														
35	A														
36	A														
37	A														
38	A														
39	A														
40	A														
41	A														
42	A														
43	A														
44	A														
45	A														
46	A														
47	A														
48	A														
49	A														
50	A														
51	A														
52	A														
53	A														
54	A														
55	A														
56	A														
57	A														
58	A														
59	A														
60	A														
61	A														
62	A														
63	A														
64	A														
65	A														
66	A														
67	A														
68	A														
69	A														
70	A														
71	A														
72	A														
73	A														
74	A														
75	A														
76	A														
77	A														
78	A														
79	A														
80	A														

*Number of sheets per pad may vary slightly.

Figure 7-1 (Part 1 of 2). Example of the Data Description Specifications for a Display Device File

This display device file contains two record formats: CUSPMT and CUSFLDS.

- 1** The attributes for the fields in this file are defined in the CUSMSTP field reference file.
- 2** Command attention key 1 is associated with indicator 15, with which the user ends the program.
- 3** The ERRMSG keyword identifies the error message that is displayed if indicator 99 is set on in the program that uses this record format.
- 4** The OVERLAY keyword is used for the record format CUSFLDS so that the CUSPMT record on the display will not be erased when the CUSFLDS record is written to the display.
- 5** The constants such as 'Name', 'Address', and 'City' describe the fields that are written out by the program.
- 6** The line and position entries identify where the fields or constants are written on the display.

Figure 7-1 (Part 2 of 2). Example of the Data Description Specifications for a Display Device File

Processing an Externally Described TRANSACTION File

When an externally described TRANSACTION file is processed, the Control Program Facility (CPF) transforms data from the program to the format specified for the file and displays the data. When data passes to the program, the data is transformed to the format used by the program.

CPF provides device control information for performing input/output operations for the device. When an input record is requested from the device, CPF issues the request, then removes device control information from the data before passing the data to the program. In addition, CPF can pass indicators to the program indicating which fields, or if any fields, in the record have changed.

When the program requests an output operation, it passes the output record to CPF. CPF provides the necessary device control information to display the record. CPF also adds any constant information specified for the record format when the record is displayed.

When a record passes to a program, the fields are arranged in the order in which they are specified in the DDS. The order in which the fields are displayed is based on the display positions (line numbers and positions) assigned to the fields in the DDS. Therefore, the order in which the fields are specified in the DDS and the order in which they appear on the screen need not be the same.

Indicators

Indicators are Boolean data items that can have the values B"0" or B"1".

When you define a record format for a file using DDS, you decide the options that are to be controlled by indicators, and the indicators that are to reflect particular responses.

Option indicators provide options such as spacing, underlining, and allowing or requesting data transfer from a program to a printer or display device. Response indicators provide response information to a program from a device, such as which function keys are pressed by a work station user, and whether data has been entered.

Indicators can also be used with FORMATFILE files.

Indicators can be passed with data records in a record area, or outside the record area in a separate indicator area.

Indicators in the Record Area

If the keyword INDARA is not used in the DDS for a file, indicators are created in the record area. When indicators are defined in a record format for a file, they are read, rewritten, and written with the data in the record area.

The number and order of indicators defined in the DDS for a record format for a file determines the number and order in which the data description entries for the indicators in the record format must be coded in the program.

If a COPY statement, DDS or DD format, is used to copy indicators into a source program, the indicators are defined in the order in which they are specified in the DDS for the file.

Indicators in a Separate Indicator Area

If the file level keyword INDARA is specified in the DDS, then for any record format in the file, all indicators defined in the record format(s) for the file are passed to and from the program in a separate indicator area, not in the record area. For information on how to specify the INDARA keyword, see the *CPF Reference Manual – DDS*.

The file control entry for a file that has INDARA specified in its DDS must have the separate indicator area attribute, SI, as part of the assignment-name in the ASSIGN clause.

The advantages of using a separate indicator area are as follows:

- The number and order of indicators used in an I-O statement for any record format in a file need not match the number and order of indicators specified in the DDS for that record format.
- The program associates the indicator number in a data description entry with the appropriate indicator.

ASSIGN Clause with Separate Indicator Area Attribute

Assignment-name of the ASSIGN clause of a file control entry has the following general format:

device – System/38 file name – attribute

Device can be either WORKSTATION or FORMATFILE.

System/38 file name must refer to a file that has the file level keyword INDARA specified in its DDS.

Attribute must be the separate indicator area attribute, SI.

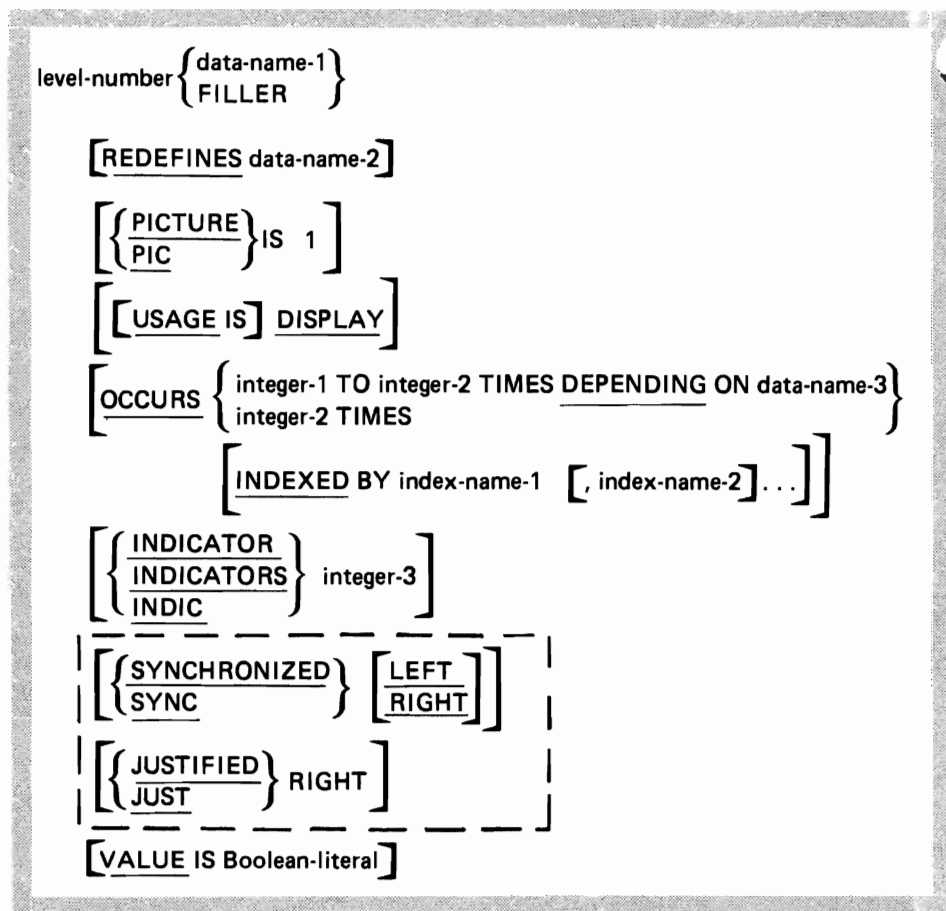
An example of an assignment-name is:

WORKSTATION – System/38 file name – SI

Data Description Entry – Boolean Data

When you use indicators in a COBOL program, you must describe them as Boolean data items using the data description entry for Boolean data.

Format



Special Considerations

The special considerations for the clauses used with the Boolean data follow. All other rules for clauses are the same as those for other data as described under "Data Description Entry" in Chapter 4.

PICTURE Clause: An elementary Boolean data-name is defined by a PICTURE containing a single 1.

USAGE Clause: USAGE must be defined implicitly or explicitly as DISPLAY.

OCCURS Clause: When the OCCURS clause and the INDICATOR clause are both specified at an elementary level, a table of Boolean data items is defined with each element in the table corresponding to an external indicator. The first element in the table corresponds to the indicator number specified in the INDICATOR clause; the second element corresponds to the indicator that sequentially follows the indicator specified by the INDICATOR clause.

For example, if the following is coded:

```
07 SWITCHES PIC 1
          OCCURS 10 TIMES
          INDICATOR 16.
```

then SWITCHES (1) corresponds to indicator 16, SWITCHES (2) corresponds to indicator 17, ..., and SWITCHES (10) corresponds to indicator 25.

INDICATOR Clause: If indicator fields are in a separate indicator area, the INDICATOR clause associates an indicator defined in DDS with a Boolean data item. If indicator fields are in the record area, the INDICATOR clause is syntax-checked, but is treated as documentation.

Integer-3 must be a value of 1 through 99.

The INDICATOR clause must be specified at an elementary level only.

VALUE Clause: The VALUE clause specifies the initial content of a Boolean data item. The allowable values for Boolean literals are B"0", B"1", and ZERO.

INDICATOR Attribute of the COPY Statement, DDS Format or DD Format

The INDICATOR attribute specifies whether or not data description entries are generated for indicators.

If the INDICATOR attribute is specified, data description entries are generated for indicators, but not for data fields. The data description entries that are generated are determined by which one of the usage attributes (I, O, or I-O) is specified or assumed in the COPY statement.

- If `...I-INDICATOR...` is specified, data description entries for input (response) indicators are generated for indicators used in the input record area.
- If `...O-INDICATOR...` is specified, data description entries for output (option) indicators are generated for indicators used in the output record area.
- If `...I-O-INDICATOR...` is specified or assumed, data description entries for both input and output (response and option) indicators are generated for indicators used in the input and output record areas.

If the INDICATOR attribute is not specified, whether data description entries are generated for indicators depends on whether the file has the keyword INDARA specified in the DDS.

- If INDARA is not specified, data description entries are generated for both data fields and indicators.
- If INDARA is specified, data description entries are generated for data fields only, not for indicators.

Group level names are assigned as follows:

- INPUT
05 format-name-I
- OUTPUT
05 format-name-O

If indicators are requested, or exist in the format, an additional group name (06 level) is generated at the beginning of the structure.

```
06 format-name-(I or O)-INDIC
07 IN01 PIC 1 INDIC 01.
07 IN04 PIC 1 INDIC 04.
07 IN05 PIC 1 INDIC 05.
07 IN07 PIC 1 INDIC 07.
06 FLD1 PIC .....
06 FLD2 PIC .....

```

If redefinition is required, a group level name is generated as follows:

```
05 file-name-RECORD
PIC X(largest record size).

```

Field names, PICTURE definitions, and numeric usage clauses are derived directly from the internal DDS format field names and data type representations. Field names and PICTURE definitions are constructed as follows:

- Data Field Structures:

06 field-name PIC (appropriate COBOL definition).

- Indicator Structures:

07 INxx PIC 1 INDIC xx.

where xx is indicator number.

See “Source Program Library” in Chapter 6 for more information on the COPY statement.

INDICATORS Phrase

When the INDICATORS phrase is used in READ, REWRITE, and WRITE statements (see Figure 7-3) it specifies which indicators are to be read, rewritten, and written.

The identifier specified in the INDICATORS phrase can be either of the following:

- An elementary Boolean data item
- A group item with elementary Boolean data items subordinate to it.

Indicators in the Record Area

If INDARA is not specified in the DDS for the file, the size of the identifier in the INDICATORS phrase of an I-O statement (see Figure 7-3) should be equal to the number of option or response indicators defined in the DDS for that format.

- In a READ statement, the identifier size should be equal to the number of response indicators.
- In a REWRITE or WRITE statement, the identifier size should be equal to the number of option indicators.

The contents of the identifier are not checked, but are copied to or from the beginning of the record, on a byte by byte basis; indicator numbers are ignored.

If the INDICATORS phrase is omitted, the data in the indicator fields in the record are still passed in the record area. The INDICATORS phrase is only used to copy indicators into the record area before a WRITE or REWRITE statement, or out of the record area after a READ statement.

Indicators in a Separate Indicator Area

If INDARA is specified in the DDS for the file, the use of the indicators referenced in the INDICATORS phrase is based on indicator number.

- In a READ statement, only the response indicator numbers referenced by the INDICATORS phrase are updated. Indicators specified in the DDS for the format but not referenced by the INDICATORS phrase are ignored. Indicators referenced by the INDICATORS phrase but not specified in the DDS are not modified.
- In a WRITE or REWRITE statement, only the option indicators referenced by the INDICATORS phrase are used. Indicators specified in the DDS for the format but not referenced by the INDICATORS phrase are assumed to be off. Indicators referenced by the INDICATORS phrase but not used in the DDS for the format are ignored.

If the INDICATORS phrase is not specified, the following occurs:

- In the READ statement, indicators are not updated.
- In a WRITE or REWRITE statement, indicators are treated as through they are set off.

Indicators Sample Programs

This section contains sample COBOL programs that illustrate the use of indicators in either a record area or a separate indicator area.

All the programs do the following:

1. Determine the current date.
2. If it is the first day of the month, turn on an option indicator that causes an output field to appear and blink.
3. Allow the work station user to press CF keys to terminate the program, or turn on response indicators and call programs to write daily or monthly reports.

Figure 7-2 shows a program that uses indicators in the record area but does not use the INDICATORS phrase in any I-O statement. The associated DDS for the file is also shown.

Figure 7-3 shows a program that uses indicators in the record area and the INDICATORS phrase in the I-O statements. The associated DDS for the file is the same as that shown for the program in Figure 7-2.

Figure 7-4 shows a program that uses indicators in a separate indicator area, defined in WORKING-STORAGE by using the COPY statement, DDS format. The associated DDS for the file is also shown.

Figure 7-5 shows a program that uses indicators in a separate indicator area, defined in a table in WORKING-STORAGE. The associated DDS for the file is the same as that shown for the program in Figure 7-4.

File		Keying Instruction	Graphic				Description	Page of
Programmer	Date		Key					

Sequence Number	Form Type And/Or Comment (A/O/*)	Conditioning				Name	Length	Reference (R)	Data Type (B/A/P/S/B/A/S/X/Y/N/I/W)	Decimal Positions	Usage (B/O//B/H/M)	Location		Functions
		Indicator	Not (N)	Indicator	Not (N)							Line	Pos	
1														
2														
3														
4														
5	*													
6	*													
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														
25														
26														
27														
28														
29														
30														
31														
32														
33														
34														
35														
36														
37														
38														
39														
40														
41														
42														
43														
44														
45														
46														
47														
48														
49														
50														
51														
52														
53														
54														
55														
56														
57														
58														
59														
60														
61														
62														
63														
64														
65														
66														
67														
68														
69														
70														
71														
72														
73														
74														
75														
76														
77														
78														
79														
80														
	A													
	A													
	A													
	A													
	A													
	A													
	A													
	A													
	A													
	A													
	A													
	A													
	A													
	A													

- 1 The INDARA keyword is not used; indicators are stored in the record area with the data fields.
- 2 One record format, FORMAT1, is specified.
- 3 Three indicators are associated with three CF keys. Indicator 99 will be set on when CF key 1 is pressed, and so on.
- 4 One field is defined for input.
- 5 Indicator 01 is defined to cause the associated constant field to blink if the indicator is on.
- 6 The CF key definitions are documented on the work station screen.

Figure 7-2 (Part 1 of 4). Example of a Program Using Indicators in the Record Area without Using the INDICATORS Phrase in the I-O Statement – Data Description Specifications

STMT SEQNBR -A 1 B.. ... 2 3 4 5 5 7 .IDENTFCN S COPYNAME

```

1 000100 IDENTIFICATION DIVISION.                                00010000
2 000200   PROGRAM-ID. INDIC1.                                  00020000
000300*   INDICATORS EXAMPLE 1 - FILE WITH INDICATORS IN RECORD AREA. *00050000
3 000400 ENVIRONMENT DIVISION.                                  00170000
4 000500   CONFIGURATION SECTION.                              00180000
5 000600   SOURCE-COMPUTER. IBM-S38.                          00190000
6 000700   OBJECT-COMPUTER. IBM-S38.                          00200000
7 000800   INPUT-OUTPUT SECTION.                              00250000
8 000900   FILE-CONTROL.                                       00260000
9 001000       SELECT DISPFILE                                 00330000
10 001100           ASSIGN TO WORKSTATION-DISPFILEX 1          00340000
11 001200           ORGANIZATION IS TRANSACTION                00350000
12 001300           ACCESS          IS SEQUENTIAL.             00350000
13 001400                                                     00360000
14 001500 DATA DIVISION.                                       00370000
15 001600   FILE SECTION.                                       00380000
16 001700   FD  DISPFILE                                       05500000
17 001800       LABEL RECORDS ARE OMITTED                     00560000
18 001900       DATA RECORD IS DISP-REC.                     05700000
19 002000   01  DISP-REC.                                       05800000
20 002100       COPY DDS-ALL-FORMATS OF DISPFILEX. 2
22 +000001           05  DISPFILEX-RECORD PIC X(8).
23 +000002                                                     <-ALL-FMTS
+000003* INPUT FORMAT:FORMAT1 FROM FILE DISPFILEX OF LIBRARY EXAMPLES <-ALL-FMTS
+000004*                                                     <-ALL-FMTS
24 +000005           05  FORMAT1-I REDEFINES DISPFILEX-RECORD. <-ALL-FMTS
25 +000006           06  FORMAT1-I-INDIC.                       <-ALL-FMTS
26 +000007               07 IN99 PIC 1 INDIC 99. 3           <-ALL-FMTS
+000008*               END OF PROGRAM                          <-ALL-FMTS
27 +000009               07 IN51 PIC 1 INDIC 51.               <-ALL-FMTS
+000010*               DAILY REPORT                           <-ALL-FMTS
28 +000011               07 IN52 PIC 1 INDIC 52.               <-ALL-FMTS
+000012*               MONTHLY REPORT                          <-ALL-FMTS
29 +000013           06  DEPTNO PIC X(5).                       <-ALL-FMTS
30 +000014 OUTPUT FORMAT:FORMAT1 FROM FILE DISPFILEX OF LIBRARY EXAMPLES <-ALL-FMTS
+000015*                                                     <-ALL-FMTS
31 +000017           05  FORMAT1-O REDEFINES DISPFILEX-RECORD. <-ALL-FMTS
32 +000018           06  FORMAT1-O-INDIC.                       <-ALL-FMTS
33 +000019               07 IN01 PIC 1 INDIC 01.               <-ALL-FMTS
34 002200                                                     00360000
35 002300 WORKING-STORAGE SECTION.                               00620000
36 002400   01  CURRENT-DATE.                                    05800000
37 002500       05  CURR-YEAR PIC 99.                            05800000
38 002600       05  CURR-MONTH PIC 99.                          05800000
39 002700       05  CURR-DAY PIC 99.                             05800000
40 002800                                                     05800000
41 002900   01  INDIC-AREA. 4                                  05800000
42 003000       05  IN01 PIC 1.                                  05800000
43 003100       5  88 NEW-MONTH PIC 1. VALUE B"1".             05800000
44 003200       05  IN51 PIC 1.                                  05800000
45 003300       88 WANT-DAILY PIC 1. VALUE B"1".               05800000
46 003400       05  IN52 PIC 1.                                  05800000
47 003500       88 WANT-MONTHLY PIC 1. VALUE B"1".             05800000
48 003600       05  IN99 PIC 1.                                  05800000

```

Figure 7-2 (Part 2 of 4). Example of a Program Using Indicators in the Record Area without Using the INDICATORS Phrase in the I-O Statement


```

5714CB1 R04 M01 820813          COBOL SOURCE LISTING          INDIC1          05/31/87
STMT SEQNBR -A 1 B.. ... 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 .IDENTFCN 5 COPYNAME

49 003700          88 NOT-END-OF-JOB          VALUE B"0".          0580000
50 003800          88 END-OF-JOB          VALUE B"1".          0580000
51 003900
52 004000 PROCEDURE DIVISION.
53 004100          OPEN I-O DISPFILE.          01030000
54 004200          ACCEPT CURRENT-DATE FROM DATE.  090000
55 004300          SET NOT-END-OF-JOB TO TRUE.
56 004400          PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
004500          UNTIL END-OF-JOB.
57 004600          CLOSE DISPFILE.
58 004700          STOP RUN.
004800
004900 DISPLAY-SCREEN.          01030000
59 005000          6 → MOVE ZEROS TO INDIC-AREA.
60 005100          7 → IF CURR-DAY = 01 THEN
61 005200          8 → SET NEW-MONTH TO TRUE.
62 005300          9 → MOVE CORR INDIC-AREA TO FORMAT1-O-INDIC.
63 005400          10 → WRITE DISP-REC FORMAT IS "FORMAT1".
005500
005600 READ-AND-PROCESS-SCREEN.
64 005700          11 → MOVE ZEROS TO INDIC-AREA.
65 005800          12 → READ DISPFILE FORMAT IS "FORMAT1".
66 005900          13 → MOVE CORR FORMAT1-I-INDIC TO INDIC-AREA.
67 006000          IF WANT-DAILY THEN
68 006100          CALL "DAILY" USING DEPTNO
006200          ELSE
69 006300          IF WANT-MONTHLY THEN
70 006400          CALL "MONTHLY" USING DEPTNO.

          * * * * * E N D O F S O U R C E * * * * *

```

Figure 7-2 (Part 3 of 4). Example of a Program Using Indicators in the Record Area without Using the INDICATORS Phrase in the I-O Statement

```

5714CB1 R04 M01 820813          COBOL SOURCE LISTING          INDIC2          05/31/82
STMT SEQNBR -A 1 B.. ... 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 .IDENTFCN S COPYNAME

49 003700          05 NEW-MONTH          PIC 1.          000
50 003800
51 003900 PROCEDURE DIVISION.          01030000
52 004000          OPEN I-O DISPFILE.          01060000
53 004100          ACCEPT CURRENT-DATE FROM DATE.          090000
54 004200          MOVE IND-OFF TO END-OF-PROGRAM.
55 004300          PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
004400          UNTIL END-OF-PROGRAM = IND-ON.
56 004500          CLOSE DISPFILE.
57 004600          STOP RUN.
004700          01030000
004800 DISPLAY-SCREEN.
58 004900          MOVE ZEROS TO OPTION-INDICS.
59 005000          IF CURR-DAY = 01 THEN
60 005100          5 MOVE IND-ON TO NEW-MONTH.
61 005200          6 WRITE DISP-REC FORMAT IS "FORMAT1"
005300          INDICATORS ARE OPTION-INDICS.
005400
005500 READ-AND-PROCESS-SCREEN.
62 005600          MOVE ZEROS TO RESPONSE-INDICS.
63 005700          7 READ DISPFILE FORMAT IS "F0RMAT1"
005800          INDICATORS ARE RESPONSE-INDICS. 8
64 005900          IF DAILY-REPORT = IND-ON THEN
65 006000          9 CALL "DAILY" USING DEPTNO
006100          ELSE
66 006200          IF MONTHLY-REPORT = IND-ON THEN
67 006300          CALL "MONTHLY" USING DEPTNO.

          * * * * * E N D   O F   S O U R C E * * * * *

```

Figure 7-3 (Part 2 of 3). Example of a Program Using Indicators in the Record Area and the INDICATORS phrase in the I-O Statements

- 1** The separate indicator area attribute, SI, is not coded in the ASSIGN clause.
- 2** The COPY statement, DDS format, defines data fields and indicators in the record area.
- 3** Because the file does not have a separate indicator area, response and option indicators are defined in the order in which they are used in the DDS, and the indicator numbers are treated as documentation.
- 4** All indicators used by the program are defined with meaningful names in data description entries in WORKING-STORAGE. Indicator numbers are omitted here because they have no effect. Indicators should be defined in the order needed by the display file.
- 5** IN01 in WORKING-STORAGE is set on if it is the first day of the month.
- 6** FORMAT1 is written to the work station screen:
 - The INDICATORS phrase causes the contents of the variable OPTION-INDICS to be copied to the beginning of the record area.
 - Data and indicator values are written to the work station screen.
- 7** FORMAT1, including both data and indicators, is read from the work station screen.
- 8** The INDICATORS phrase causes bytes to be copied from the beginning of the record area to RESPONSE-INDICS.
- 9** If CF key 5 has been pressed, a program call is executed.

Figure 7-3 (Part 3 of 3). Example of a Program Using Indicators in the Record Area and the INDICATORS phrase in the I-O Statements

File	Keying Instruction	Graphic							
Programmer	Date	Key							

Description	Page	of
-------------	------	----

Sequence Number	Form Type And/Or Comment (A/O/*)	Conditioning				Name	Length	Reference (R)	Location		Functions
		Indicator	Not (N)	Indicator	Not (N)				Line	Pos	
A*					DISPLAY FILE DPS						FOR INDICATOR EXAMPLES
A*											INDARA
A					R FORMAT I						CF01(99 'END OF PROGRAM')
A											CF05(51 'DAILY REPORT')
A											CF09(52 'MONTHLY REPORT')
A*											
A									10	10	'DEPARTMENT NUMBER: '
A					DEPTNO	5			I 10	32	
A		01							20	26	'PRODUCE MONTHLY REPORTS'
A*											DSPATR(BL)
A									24	01	'CFS = DAILY REPORT'
A									24	26	'CF9 = MONTHLY REPORT'
A									24	53	'CF1 = TERMINATE'
A											
A											

1 The INDARA keyword is specified, indicators are stored in a separate indicator area, not in the record area. Except for this specification, the DDS for this file is the same as that shown in Figure 7-2.

Figure 7-4 (Part 1 of 4). Example of a Program Using Indicators in a Separate Indicator Area, defined in WORKING-STORAGE by using the COPY Statement, DDS format

```

STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S COPYNAME

 1 000100 IDENTIFICATION DIVISION.                                00010000
 2 000200 PROGRAM-ID. INDIC3.                                    00020000
 3 000300* INDICATORS EXAMPLE 3 - FILE WITH SEPARATE INDICATOR AREA. *00050000
 4 000400 ENVIRONMENT DIVISION.                                  00170000
 5 000500 CONFIGURATION SECTION.                                00180000
 6 000600 SOURCE-COMPUTER. IBM-S38.                             00190000
 7 000700 OBJECT-COMPUTER. IBM-S38.                             00200000
 8 000800 INPUT-OUTPUT SECTION.                                 00250000
 9 000900 FILE-CONTROL.                                         00260000
10 001000 SELECT DISPFIL                                         00330000
11 001100 ASSIGN TO WORKSTATION-DISPFIL-SI 1                    00340000
12 001200 ORGANIZATION IS TRANSACTION                           00350000
13 001300 ACCESS IS SEQUENTIAL.                                 00350000
14 001400                                                         00360000
15 001500 DATA DIVISION.                                       00370000
16 001600 FILE SECTION.                                         00380000
17 001700 FD DISPFIL                                           05500000
18 001800 LABEL RECORDS ARE OMITTED                             00560000
19 001900 DATA RECORD IS DISP-REC.                             05700000
20 002000 01 DISP-REC.                                          05800000
21 002100 COPY DDS-ALL-FORMATS OF DISPFIL. 2
22 +000001 05 DISPFIL-RECORD PIC X(8).                          <-ALL-FMTS
23 +000002                                                         <-ALL-FMTS
24 +000003* INPUT FORMAT:FORMAT1 FROM FILE DISPFIL OF LIBRARY EXAMPLES <-ALL-FMTS
25 +000004*                                                         <-ALL-FMTS
26 +000005 05 FORMAT1-I REDEFINES DISPFIL-RECORD.               <-ALL-FMTS
27 +000006 06 DEPTNO PIC X(5).                                  <-ALL-FMTS
28 +000007                                                         <-ALL-FMTS
29 +000008* OUTPUT FORMAT:FORMAT1 FROM FILE DISPFIL OF LIBRARY EXAMPLES <-ALL-FMTS
30 +000009*                                                         <-ALL-FMTS
31 +000010* 05 FORMAT1-O REDEFINES DISPFIL-RECORD.              <-ALL-FMTS
32 002200                                                         00360000
33 002300 WORKING-STORAGE SECTION.                               00620000
34 002400 01 CURRENT-DATE.                                       05800000
35 002500 05 CURR-YEAR PIC 99.                                    05800000
36 002600 05 CURR-MONTH PIC 99.                                  05800000
37 002700 05 CURR-DAY PIC 99.                                    05800000
38 002800                                                         05800000
39 002900 77 IND-OFF PIC 1 VALUE B"0".                           05800000
40 003000 77 IND-ON PIC 1 VALUE B"1".                             05800000
41 003100 01 DISPFIL-INDICS.                                     05800000
42 003200 COPY DDS-ALL-FORMATS-INDIC OF DISPFIL. 3              05800000
43 +000001                                                         <-ALL-FMTS
44 +000002* INPUT FORMAT:FORMAT1 FROM FILE DISPFIL OF LIBRARY EXAMPLES <-ALL-FMTS
45 +000003*                                                         <-ALL-FMTS
46 +000004 06 FORMAT1-I-INDIC.                                    <-ALL-FMTS
47 +000005 07 IN51 PIC 1 INDIC 51. 4                               <-ALL-FMTS
48 +000006* DAILY REPORT                                         <-ALL-FMTS
49 +000007 07 IN52 PIC 1 INDIC 52.                                <-ALL-FMTS
50 +000008* MONTHLY REPORT                                       <-ALL-FMTS
51 +000009 07 IN99 PIC 1 INDIC 99.                                <-ALL-FMTS
52 +000010* END OF PROGRAM                                       <-ALL-FMTS
53 +000011                                                         <-ALL-FMTS
54 +000012* OUTPUT FORMAT:FORMAT1 FROM FILE DISPFIL OF LIBRARY EXAMPLES <-ALL-FMTS
55 +000013*                                                         <-ALL-FMTS

```

Figure 7-4 (Part 2 of 4). Example of a Program Using Indicators in a Separate Indicator Area, defined in WORKING-STORAGE by using the COPY Statement, DDS format

```

5714CB1 R04 M01 820813          COBOL SOURCE LISTING          INDIC3          05/31/82
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S COPYNAME

45 +000014          06 FORMAT1-O-INDIC.          <-ALL-FMTS
46 +000015          07 IN01          PIC 1  INDIC 01.          <-ALL-FMTS
47 003300          01030000
48 003400 PROCEDURE DIVISION.          01060000
49 003500          OPEN I-O  DISPFILE.          090000
50 003600          ACCEPT CURRENT-DATE FROM DATE.
51 003700          MOVE IND-OFF TO IN99 IN FORMAT1-I-INDIC.
52 003800          PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
003900          UNTIL IN99 IN FORMAT1-I-INDIC = IND-ON.
53 004000          CLOSE DISPFILE.
54 004100          STOP RUN.
004200          01030000
004300 DISPLAY-SCREEN.
55 004400          MOVE ZEROS TO FORMAT1-O-INDIC.
56 004500          IF CURR-DAY = 01 THEN
57 004600          MOVE IND-ON TO IN01 IN FORMAT1-O-INDIC.
58 004700          WRITE DISP-REC  FORMAT IS "FORMAT1"
004800          INDICATORS ARE FORMAT1-O-INDIC.
004900          1260000
005000 READ-AND-PROCESS-SCREEN.
59 005100          MOVE ZEROS TO FORMAT1-I-INDIC.
60 005200          READ DISPFILE  FORMAT IS "FORMAT1"
005300          INDICATORS ARE FORMAT1-I-INDIC.
61 005400          IF IN51 IN FORMAT1-I-INDIC = IND-ON THEN
62 005500          CALL "DAILY" USING DEPTNO
005600          ELSE
63 005700          IF IN52 IN FORMAT1-I-INDIC = IND-ON THEN
64 005800          CALL "MONTHLY" USING DEPTNO.
          1260000

          * * * * *  E N D  O F  S O U R C E  * * * * *

```

Figure 7-4 (Part 3 of 4). Example of a Program Using Indicators in a Separate Indicator Area, defined in WORKING-STORAGE by using the COPY Statement, DDS format

- 1** The separate indicator area attribute, SI, is specified in the ASSIGN clause.
- 2** The COPY statement, DDS format, generates data descriptions in the record area for data fields only. The data description entries for the indicators are not generated because a separate indicator area has been specified for the file.
- 3** The COPY statement, DDS format, with the INDICATOR attribute, INDIC, defines data description entries in WORKING-STORAGE for all indicators used in the DDS for the record format for the file.
- 4** Because the file has a separate indicator area, the indicator numbers used in the data description entries are not treated as documentation.
- 5** IN01 in the separate indicator area for FORMAT1 is set on if it is the first day of the month.
- 6** The INDICATORS phrase is required to send indicator values to the work station screen.
- 7** The INDICATORS phrase is required to receive indicator values from the work station screen. If CF key 5 has been pressed, IN51 is set on.
- 8** If IN51 has been set on, a program call is executed.

Figure 7-4 (Part 4 of 4). Example of a Program Using Indicators in a Separate Indicator Area, defined in WORKING-STORAGE by using the COPY Statement, DDS format

STMT	SEQNBR	-A	1	B...	2	...	3	...	4	...	5	...	6	...	7	.IDENTFCN	5	COPYNAME
1	000100																	00010000
2	000200																	00020000
	000300*																	*00050000
3	000400																	00170000
4	000500																	00180000
5	000600																	00190000
6	000700																	00200000
7	000800																	00250000
8	000900																	00260000
9	001000																	00330000
10	001100																	00340000
11	001200																	00350000
12	001300																	00350000
13	001400																	00360000
14	001500																	00370000
15	001600																	00380000
16	001700																	05500000
17	001800																	00560000
18	001900																	05700000
19	002000																	05800000
20	002100																	
22	+000001																	
23	+000002																	
	+000003*																	
	+000004*																	
24	+000005																	
25	+000006																	
26	+000007																	
	+000008*																	
	+000009*																	
	+000010*																	
27	002200																	00360000
28	002300																	00620000
29	002400																	05800000
30	002500																	05800000
31	002600																	05800000
32	002700																	05800000
33	002800																	05800000
34	002900																	05800000
35	003000																	05800000
36	003100																	05800000
37	003200																	05800000
38	003300																	05800000
39	003400																	05800000
40	003500																	05800000
41	003600																	05800000
42	003700																	05800000
43	003800																	05800000
44	003900																	01030000
45	004000																	01060000
46	004100																	090000
47	004200																	
48	004300																	
49	004400																	
	004500																	

<-ALL-FMTS
 <-ALL-FMTS
 <-ALL-FMTS
 <-ALL-FMTS
 <-ALL-FMTS
 <-ALL-FMTS
 <-ALL-FMTS
 <-ALL-FMTS
 <-ALL-FMTS
 <-ALL-FMTS

Figure 7-5 (Part 1 of 2). Example of a Program Using Indicators in a Separate Indicator Area, Defined in a Table in WORKING-STORAGE

```

5714CB1 R04 M01 820813          COBOL SOURCE LISTING          INDIC4          05/31/82
STMT SEQNBR -A 1 B.. ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME

50 004600          CLOSE DISPFILE.
51 004700          STOP RUN.
   004800
   004900 DISPLAY-SCREEN.          01030000
52 005000          MOVE ZEROS TO INDIC-AREA.
53 005100          IF CURR-DAY = 01 THEN
54 005200             SET IND-ON (IND-NEW-MONTH) TO TRUE. 5
55 005300          WRITE DISP-REC  FORMAT IS "FORMAT1"
   005400             INDICATORS ARE INDIC-TABLE. 6          1260000
   005500
   005600 READ-AND-PROCESS-SCREEN.
56 005700          READ DISPFILE  FORMAT IS "FORMAT1"
   005800             INDICATORS ARE INDIC-TABLE. 7          1260000
57 005900 8 IF IND-ON (IND-DAILY) THEN
58 006000          CALL "DAILY" USING DEPTNO
   006100          ELSE
59 006200             IF IND-ON (IND-MONTHLY) THEN
60 006300             CALL "MONTHLY" USING DEPTNO.

          * * * * * E N D   O F   S O U R C E * * * * *

```

- 1 The separate indicator area attribute, SI, is specified in the ASSIGN clause.
- 2 The COPY statement, DDS format, generates fields in the record area for data fields only.
- 3 A table of 99 Boolean data items is defined in WORKING-STORAGE. The INDICATOR clause for this data description entry causes these data items to be associated with indicators 1 through 99 respectively. The use of such a table may result in improved performance as compared to the use of a group item with multiple subordinate entries for individual indicators; however, you must consider the number of references and indicators for example, to realize improved performance.
- 4 A series of data items is defined in WORKING-STORAGE to provide meaningful subscript names with which to refer to the table of indicators. The use of such data items is not required.
- 5 INDIC-TABLE (01) in the separate indicator area for FORMAT1 is set on if it is the first day of the month.
- 6 The INDICATOR phrase is required to send indicator values to the work station screen.
- 7 The INDICATOR phrase is required to receive indicator values from the work station screen. If CF key 5 has been pressed, INDIC-TABLE (51) is set on.
- 8 If INDIC-TABLE (51) has been set on, a program call is executed.

Figure 7-5 (Part 2 of 2). Example of a Program Using Indicators in a Separate Indicator Area, Defined in a Table in WORKING-STORAGE

Subfiles

Subfiles can be specified in the DDS for a display file or mixed file to allow the user to handle multiple records of the same type on a display (see Figure 7-6). A subfile is a group of records that is read from or written to a display device. For example, a program reads records from a data base file and creates a subfile of output records. When the entire subfile has been written, the program sends the entire subfile to a display device in one write operation. The work station user can change data or enter additional data in the subfile; the program then reads the entire subfile from the

display device into the program and processes each record in the subfile individually.

Records to be included in a subfile are specified in the DDS for the file. The number of records that can be contained in a subfile must also be specified in the DDS. One file can contain more than one subfile; however, only twelve subfiles can be active concurrently for a device. Twelve subfiles can be displayed on a device at the same time.

The DDS for a subfile consists of two record formats: a subfile record format and a subfile control record format. The subfile record format contains the field descriptions of all the records in the subfile. Specification of the subfile control record format on the READ or WRITE statement causes the physical read, write, or setup operations of a subfile to take place.

Figure 7-7 shows an example of the DDS for a subfile record format, and Figure 7-8 shows an example of the DDS for a subfile control record format.

For a description of how the records in a subfile can be displayed and for a description of the keywords that can be specified for a subfile, see the *CPF Programmer's Guide*.

Programming Note: In a mixed file, the formats used for defining the subfile records and the subfile control record can be used only in I-O operations to display devices.

Customer Name Search				
Search Code _____				
Number	Name	Address	City	State
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX
XXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX

Figure 7-6. Subfile Display

To use a subfile for a display file or mixed file in a COBOL program, the SUBFILE phrase must be specified with the input/output operation. The valid subfile operations are:

- READ SUBFILE file-name RECORD

- WRITE SUBFILE record-name
- REWRITE SUBFILE record-name.

In COBOL, subfiles can be processed sequentially with the READ SUBFILE NEXT MODIFIED statement, or processed randomly by specifying a relative key value.

The TRANSACTION file must be an externally described file. In COBOL, all access to the subfile is done with a relative record number. If the SUBFILE phrases are used with a TRANSACTION file, the SELECT statement in the Environment Division must state that ACCESS MODE IS DYNAMIC and must specify the RELATIVE KEY to be used.

If more than one display device is acquired by a display file or mixed file, there is a separate subfile for each individual display device. If a subfile has been created for a particular display device acquired by a TRANSACTION file, all input operations that refer to a record format for the subfile are performed against the subfile belonging to that device. See the discussions on the TERMINAL phrase later in this chapter for information about how to determine which device is used. Any operations that reference a record format name that is not designated as a subfile are executed as an input/output operation directly to the display device.

Use of Subfiles

Some typical uses of subfiles include:

- Display only. The work station user reviews the display.
- Display with selection. The user requests more information about one of the items on the display.
- Modification. The user modifies one or more of the records.
- Input only, with no validity checking. A subfile is used for a data entry function.
- Input only, with validity checking. A subfile is used for a data entry function, but the records are checked.
- Combination of tasks. A subfile can be used as a display with modification.

DATA DESCRIPTION SPECIFICATIONS

File	Keying Instruction	Graphic	Key	Description	Page	of
Programmer						

Sequence Number	Form Type 7 8 9	And/Or Comment (A/O/) 10 11 12	Conditioning				Name	Length	Reference (R)	Location		Functions
			Indicator	Not (N)	Indicator	Not (N)				Line	Pos	
		Condition Name										
		Name Type (M/R/K/S/O)										
		Reserved										
A*						DDS FOR THE DISPLAY DEVICE FILE						
A*						ACCOUNTS RECEIVABLE INTERACTIVE PAYMENT UPDATE						
A*						R SUBFILE 1				1	SFL	
A											TEXT('Subfile for customer payment')	
A*						ACPPMT	4A			5	4TEXT('Accept payment')	
A											VALUES('YES' '#NO')	
A		51									DSPATR(RI MDT)	
A		N51									DSPATR(ND PR)	
A*												
A						CUST	5			8	15TEXT('Customer number')	
A		52									DSPATR(RI)	
A		53									DSPATR(ND)	
A		54									DSPATR(PR)	
A*												
A						AMPAID	8 02B			5	24TEXT('Amount paid')	
A											CHECK(FE)	
A											AUTO(RAB)	
A											CMP(GT 0)	
A		52									DSPATR(RI)	
A		53									DSPATR(ND)	
A		54									DSPATR(PR)	
A*												
A						ECPMSG	31A			0	5	37TEXT('Exception message')
A		52									DSPATR(RI)	
A		53									DSPATR(ND)	
A		54									DSPATR(BL)	
A*												
A						OVRPMT	8Y			20	5	70TEXT('Over payment')
A												EDTCDE(L)
A		55										DSPATR(BL)
A		56										DSPATR(ND)
A*												
A						STSCDE	1A				H	TEXT('Status code')

*Number of sheets per pad may vary slightly.

Figure 7-7 (Part 1 of 2). Data Description Specifications for a Subfile Record Format

The data description specifications (DDS) for a subfile record format describe the records in the subfile:

- 1** The SFL keyword identifies the record format as a subfile.
- 2** The line and position entries define the location of the fields on the display.
- 3** The VALUES keyword specifies that the user can only specify *YES or *NO as values for the ACPMT field.
- 4** The usage entries define whether the named field is to be an output (O), input (I), output/input (B), or hidden (H) field.
- 5** The entry CHECK(FE) specifies that the user cannot skip to the next input field without pressing one of the field exit keys.
- 6** The entry AUTO(RAB) specifies that data entered into the field AMPAID is to be automatically right-justified, and the leading characters are to be filled with blanks.
- 7** The entry CMP(GT 0) specifies that the data entered for the field AMPAID is to be compared to zero to ensure that the value is greater than zero.
- 8** The EDTCDE keyword specifies the desired editing for output field OVRPMT. EDTCDE(1) indicates that the field OVRPMT is to be printed with commas, decimal point, and no sign. Also, a zero balance will be printed and leading zeros will be suppressed.
- 9** The DSPATR keyword is used to specify the display attributes for the named field when the corresponding indicator status is true. The attributes specified are BL (blink), RI (reverse image), PR (protect), MDT (set modified data tag), and ND (nondisplay).

Figure 7-7 (Part 2 of 2). Data Description Specifications for a Subfile Record Format

File	Keying Instruction	Graphic								Description	Page	of
Programmer	Date	Instruction	Key									

Sequence Number	Form Type And/OR Comment (A/O/?)	Conditioning				Name	Length	Reference (R)	Data Type (A/P/S/B/A/S/X/Y/N/I/W)	Decimal Positions	Usage (I/O/I/B/H/M)	Location		Functions
		Indicator	Not (N)	Indicator	Not (N)							Line	Pos	
A														
					R	CONTROL								TEXT('Subfile control')
														SFLCTL(SUBFILEL) 1
														SFLSIZ(17) 2
														SFLPAG(17) 3
		61												SFLCLR 4
		62												SFLDSP 5
		62												SFLDSPCTL 6
														OVERLAY
														LOCK 7
														HELP(99 'Help key') 8
														CALL(98 'End payment update')
														CALL(97 'Ignore input')
		99												9 SFLMSG(1 CF11 - Ignore invalid input + CF12 - end payment + update')
														1 2'Customer Payment Update Prompt'
														1 65'Date'
														1 71DATE EDTCDE(Y)
		63												3 2'Accept'
		63												4 2'Payment'
														3 14'Customer'
														3 26'Payment'
		64												3 37'Exception message'

*Number of sheets per pad may vary slightly.

Figure 7-8 (Part 1 of 2). Data Description Specifications for a Subfile Control Record Format

The subfile control record format defines the attributes of the subfile, the search input field, constants, and command keys. The keywords used indicate the following:

- 1** SFLCTL identifies this record as a subfile control record and names the associated subfile record (SUBFILE1).
- 2** SFLSIZ indicates the total number of records to be included in the subfile (17).
- 3** SFLPAG indicates the total number of records in a page (17).
- 4** SFLCLR indicates when the subfile should be cleared (when indicator 61 is on).
- 5** SFLDSP indicates when to display the subfile (when indicator 62 is on).
- 6** SFLDSPCTL indicates when to display the subfile control record (when indicator 62 is on).
- 7** The LOCK keyword prevents the work station user from using the keyboard when the CONTROL1 record format is initially displayed.
- 8** HELP allows the user to press the Help key and sets indicator 99 on.
- 9** SFLMSG identifies the constant as a message that is displayed if indicator 99 is on.

In addition to the control information, the subfile control record format also defines the constants to be used as column headings for the subfile record format.

Figure 7-8 (Part 2 of 2). Data Description Specifications for a Subfile Control Record Format

Multiple Device Files and Single Device Files

A multiple device file is either a display file or a mixed file capable of having more than one program device acquired for that file. Communications and BSC files can never be defined as multiple device files.

A display file is capable of having multiple program devices when the MAXDEV parameter of the CRTDSPF CL command is greater than 1. For more information about how to create and use a display file, see the *CPF Programmer's Guide*.

A mixed file is capable of having multiple program devices when the MAXPGMDEV parameter of the CRTMXDF CL command is greater than 1. For more information about how to create and use a mixed file, see the *CPF Programmer's Guide*.

COBOL determines at execution time whether a file is a single device file or a multiple device file based on whether the file is *capable* of having multiple devices. The actual number of devices acquired does not affect whether a file is considered a single or multiple device file. Determination of whether a file is a single or a multiple device file is *not* done at compilation time based on the current description of the display or mixed file.

For multiple device files, if a particular program device is to be used in an I-O statement, that device is specified by the TERMINAL phrase. The TERMINAL phrase can also be specified for a single device file.

The following example illustrates the use of multiple device files. The program uses a mixed file, and is intended to be run in batch mode. The program acquires terminals and invites those terminals via a sign-on screen. After the terminals are invited, they are polled. If nobody signs on before the wait time expires, the program ends. If the user enters a valid password, he is allowed to update an employee file by calling another COBOL program. Once the update is complete, the device is invited again and the terminals are polled again.

File		Keying Instruction		Graphic Key		Description		Page of	
Programmer	Date								
<p>A</p>									
Sequence Number	Form Type And/Or Comment (A/O/*)	Conditioning Condition Name	Name	Length	Reference (R)	Location Line Pos	Functions		
1-5	6-8 9-11 12-14 15-16 17-18	Indicator Not (N) Indicator Not (N) Indicator Not (N)	Name Type (B/R/K/S/O) Reserved			Data Type (B/A/P/S/B/F/A/S/X/Y/N/I/W) Decimal Positions Usage (B/O/I/B/H/M/P)			
1	*	DDS FOR THE MIXED FILE MULT							
2	*								
3			R SIGNON				INVITE 1		
4						0 5 20	'BBBBBBBBBBBBBBBBBBBB'		
5						0 6 20	DSPATR(RI)		
6						0 6 38	'BB'		
7						0 7 20	DSPATR(RI)		
8						0 7 27	MDDFF'		
9						0 7 38	DSPATR(HI BL)		
10						0 8 20	'BB'		
11						0 8 38	DSPATR(RI)		
12						0 9 20	'BBBBBBBBBBBBBBBBBBBB'		
13						0 20 20	PLEASE LOGON'		
14			PASSWORD	10A	I	20 43	DSPATR(RI PC)		
15				20A	O	21 43			
16			R UPDATE						
17						0 3	5'UPDATE OF PERSONNEL FILE'		
18						0 7	DSPATR(BL)		
19						0 7	5'TYPE IN EMPLOYEE NUMBER +		
20							TO BE UPDATED'		
21			NUM	7A	I	7 44	DSPATR(RI PC)		
22			R EMPLOYEE						
23						0 3	5'EMPLOYEE NUMBER'		
24			NUM	7A	B	3 25	DSPATR(PC)		
25						0 5	5'EMPLOYEE NAME'		
26			NAME	30A	B	5 25	DSPATR(PC)		
27						0 7	5'EMPLOYEE ADDRESS'		
28						0 9	5'STREET'		
29			STREET	30A	B	9 25	DSPATR(PC)		
30						0 11	5'APARTMENT NUMBER'		
31			APTNO	5A	B	11 25	DSPATR(PC)		
32						0 13	5'CITY'		
33			CITY	20A	B	13 25	DSPATR(PC)		
34						0 15	5'PROVINCE'		
35			PROV	20A	B	15 25	DSPATR(PC)		
36			R RECOVERY						
37						0 3	5'THE EMPLOYEE NUMBER YOU +		
38						0 6	HAVE GIVEN IS INVALID'		
39						0 6	5'TYPE Y TO RETRY'		
40						0 8	5'TYPE N TO EXIT'		
41			ANSWER	1X	I	10	5DSPATR(RI PC)		
42							VALUES('Y' & 'N')		

1 The format SIGNON has the keyword INVITE associated with it. This means that, if format SIGNON is used in a WRITE statement, the device to which it is writing will be invited.

*Number of sheets per pad may vary slightly.

Figure 7-9 (Part 1 of 10). Example of the Use of Multiple Device Files

IBM International Business Machines Corporation		DATA DESCRIPTION SPECIFICATIONS												GX21-7754-2 UM/050*		Printed in U.S.A.					
File		Keying Instruction		Graphic Key														Description		Page of	
Programmer		Date																			
A																					
Sequence Number	Form Type	Conditioning						Name	Length	Reference (R)	Data Type (B/A/P/S/B/F/A/S/X/Y/N/I/W)	Decimal Positions	Usage (B/O//B/H/M/P)	Location		Functions					
		Acc./Oz./Comment (A/O/*)	Indicator	Indicator	Indicator	Indicator	Indicator							Line	Pos						
1																					
	A	*																			
	A	*					DDS FOR THE PHYSICAL FILE PASSWORD														
	A	*																			
	A																	UNIQUE			
	A						R	PASSWORDS													
	A							PASSKEY													
	A							PASSKEY													
	A							PASSKEY													
	A																				
	A																				
	A																				
	A																				
	A																				
	A																				
	A																				
	A																				
	A																				
	A																				

Figure 7-9 (Part 2 of 10). Example of the Use of Multiple Device Files

File	Keving Instruction	Graphic	Description	Page of
Programmer	Date	Key		

Sequence Number	Form Type And/Or Comment (A/O/*)	Conditioning				Name	Length	Reference (R)	Data Type (B/A/I/S/B/F A/S/X/Y/N/I/W)	Decimal Positions	Usage (B/O/I/B/H/W/P)	Location		Functions
		Indicator	Not (N)	Indicator	Not (N)							Line	Pos	
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														
25														
26														
27														
28														
29														
30														
31														
32														
33														
34														
35														
36														
37														
38														
39														
40														
41														
42														
43														
44														
45														
46														
47														
48														
49														
50														
51														
52														
53														
54														
55														
56														
57														
58														
59														
60														
61														
62														
63														
64														
65														
66														
67														
68														
69														
70														
71														
72														
73														
74														
75														
76														
77														
78														
79														
80														

*Number of sheets per pad may vary slightly.

Figure 7-9 (Part 3 of 10). Example of the Use of Multiple Device Files

STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN 5 COPYNAME

```

1 000100 PROCESS
2 000200 IDENTIFICATION DIVISION.
3 000300 PROGRAM-ID. SAMPLEMDF.
4 000400 AUTHOR.      A N Y PROGRAMMER.
000500*****
000600* THE FOLLOWING PROGRAM DEMONSTRATES SOME OF THE FUNCTIONS *
000700* AVAILABLE WITH MULTIPLE DEVICE FILE SUPPORT *
000800*****
5 000900 ENVIRONMENT DIVISION.
6 001000 CONFIGURATION SECTION.
7 001100 SOURCE-COMPUTER. IBM-S38.
8 001200 OBJECT-COMPUTER. IBM-S38.
9 001300 SPECIAL-NAMES.
10 001400     ATTRIBUTE-DATA IS ATTR. 1
11 001500 INPUT-OUTPUT SECTION.
12 001600 FILE-CONTROL.
13 001700     SELECT MULTIPLE-FILE
14 001800     ASSIGN TO WORKSTATION-MULT
15 001900     ORGANIZATION IS TRANSACTION 2
16 002000     ACCESS MODE IS SEQUENTIAL
17 002100     FILE STATUS IS MULTIPLE-FS1, MULTIPLE-FS2 3
18 002200     CONTROL-AREA IS MULTIPLE-CONTROL-AREA.
19 002300
20 002400     SELECT TERMINAL-FILE 4
21 002500     ASSIGN TO DATABASE-TERM
22 002600     ORGANIZATION IS SEQUENTIAL
23 002700     ACCESS MODE IS SEQUENTIAL
24 002800     FILE STATUS IS TERMINAL-FS1.
25 002900
26 003000     SELECT PASSWORD-FILE
27 003100     ASSIGN TO DATABASE-PASSWORD
28 003200     ORGANIZATION IS INDEXED
29 003300     RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
30 003400     ACCESS MODE IS RANDOM
31 003500     FILE STATUS IS PASSWORD-FS1.
32 003600
33 003700     SELECT PRINTER-FILE
34 003800     ASSIGN TO PRINTER-QPRINT.
35 003900
36 004000
37 004100 DATA DIVISION.
38 004200 FILE SECTION.
39 004300 FD MULTIPLE-FILE.
40 004400 01 MULTIPLE-REC. COPY DDS-SIGNON OF MULT. 5
42 +000001 05 MULT-RECORD PIC X(30).
43 +000002                                     SIGNON
+000003* INPUT FORMAT:SIGNON FROM FILE MULT OF LIBRARY BATCHMDF SIGNON
+000004*                                     SIGNON
44 +000005 05 SIGNON-I REDEFINES MULT-RECORD. SIGNON
45 +000006 06 PASSWORD PIC X(10). 6 SIGNON
46 +000007                                     SIGNON
+000008* OUTPUT FORMAT:SIGNON FROM FILE MULT OF LIBRARY BATCHMDF SIGNON
+000009*                                     SIGNON
47 +000010 05 SIGNON-D REDEFINES MULT-RECORD. SIGNON
48 +000011 06 FILLER PIC X(10). SIGNON

```

Figure 7-9 (Part 4 of 10). Example of the Use of Multiple Device Files

```

5714C81 R05 M00 830610          COBOL SOURCE LISTING          SAMPLEMDF          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME

49 *000012          06 WRONG          PIC X(20).          SIGNON
50 004500          FD TERMINAL-FILE.
51 004600          01 TERMINAL-FILE.
52 004700          01 TERMINAL-REC. COPY DDS-ALL-FORMATS OF TERM.
54 *000001          05 TERM-RECORD PIC X(10).          <-ALL-FMTS
55 *000002          I-O FORMAT:TERM          FROM FILE TERM          OF LIBRARY BATCHMDF          <-ALL-FMTS
56 *000003          05 TERM          REDEFINES TERM-RECORD.          <-ALL-FMTS
57 *000004          06 TERM          PIC X(10).          <-ALL-FMTS
58 004800          FD PASSWORD-FILE.
59 004900          01 PASSWORD-REC. COPY DDS-ALL-FORMATS OF PASSWRD.          <-ALL-FMTS
60 005000          05 PASSWORD-RECORD PIC X(20).          <-ALL-FMTS
62 *000001          I-O FORMAT:PASSWORDS          FROM FILE PASSWORD          OF LIBRARY BATCHMDF          <-ALL-FMTS
63 *000002          05 PASSWORDS          REDEFINES PASSWORD-RECORD.          <-ALL-FMTS
64 *000003          06 PASSKEY          PIC X(10).          <-ALL-FMTS
65 *000004          06 PASSWORD          PIC X(10).          <-ALL-FMTS
66 *000005          06 PASSKEY          PIC X(10).          <-ALL-FMTS
67 005100          FD PRINTER-FILE.
68 005200          01 PRINTER-REC.
69 005300          05 PRINTER-RECORD          PIC X(132).
70 005400          WORKING-STORAGE SECTION.
71 005500
72 005600          DECLARE THE FILE STATUS FOR EACH FILE
73 005700          *****
74 006100          01 MULTIPLE-FS1          PIC X(2) VALUE SPACES.
75 006200          01 MULTIPLE-FS2          PIC X(2) VALUE SPACES.
76 006300          05 MULTIPLE-MAJOR          PIC X(2) VALUE SPACES.
77 006400          05 MULTIPLE-MINOR          PIC X(2) VALUE SPACES.
78 006500          01 TERMINAL-FS1          PIC X(2) VALUE SPACES.
79 006600          01 PASSWORD-FS1          PIC X(2) VALUE SPACES.
80 006700          01 MULTIPLE-FS3          PIC X(2) VALUE SPACES.
81 006800          01 MULTIPLE-FS4          PIC X(2) VALUE SPACES.
82 006900          DECLARE STRUCTURE FOR HOLDING FILE ATTRIBUTES
83 007000          *****
84 007100          01 STATION-ATTR.
85 007200          05 STATION-TYPE          PIC X.
86 007300          05 STATION-SIZE          PIC X.
87 007400          05 STATION-LDC          PIC X.
88 007500          05 FILLER          PIC X.
89 007600          05 STATION-ACQUIRE          PIC X.
90 007700          05 STATION-INVITE          PIC X.
91 007800          05 STATION-DATA          PIC X.
92 007900          05 STATION-STATUS          PIC X.

```

Figure 7-9 (Part 5 of 10). Example of the Use of Multiple Device Files

```

93 008300 05 STATION-DISPLAY PIC X.
94 008400 05 STATION-KEYBOARD PIC X.
95 008500 05 STATION-SIGNON PIC X.
96 008600 05 FILLER PIC X(15).
97 008700
008800*****
008900* DECLARE THE CONTROL AREA FOR MULTIPLE-FILF *
009000*****
98 009100
99 009200 01 MULTIPLE-CONTROL-AREA.
100 009300 05 MULTIPLE-KEY-FEEDBACK PIC X(2) VALUE SPACES.
101 009400 05 MULTIPLE-DEVICE-NAME PIC X(10) VALUE SPACES.
102 009500 05 MULTIPLE-FORMAT-NAME PIC X(10) VALUE SPACES.
103 009600
009700*****
009800* DFCLARE ERROR REPORT VARIABLES *
009900*****
104 010000
105 010100 01 HEADER-LINE.
106 010200 05 FILLER PIC X(60) VALUE SPACES.
107 010300 05 FILLER PIC X(72) VALUE
108 010400 "MDF ERROR REPORT".
109 010500
110 010600 01 DETAIL-LINE.
111 010700 05 FILLER PIC X(15) VALUE SPACES.
112 010800 05 DESCRIPTION PIC X(25) VALUE SPACES.
113 010900 05 DETAIL-VALUE PIC X(92) VALUE SPACES.
114 011000
011100*****
011200* DECLARE COUNTERS, FLAGS AND STORAGE VARIABLES *
011300*****
115 011400
116 011500 01 CURRENT-TERMINAL PIC X(10) VALUE SPACES.
117 011600
118 011700 01 TERMINAL-ARRAY.
119 011800 05 LIST-OF-TERMINALS OCCURS 250 TIMES.
120 011900 07 DEVICE-NAME PIC X(10).
121 012000
122 012100 01 COUNTER PIC 999 VALUE IS 1.
123 012200
124 012300 01 NO-OF-TERMINALS PIC 999 VALUE IS 1.
125 012400
126 012500 01 TERMINAL-LIST-FLAG PIC 1.
127 012600 88 END-OF-TERMINAL-LIST VALUE IS "1".
128 012700 86 NOT-END-OF-TERMINAL-LIST VALUE IS "0".
129 012800
130 012900 01 NO-DATA-FLAG PIC 1.
131 013000 88 NO-DATA-AVAILABLE VALUE IS "1".
132 013100 88 DATA-AVAILABLE VALUE IS "0".
133 013200
134 013300 PROCEDURE DIVISION.
013400
013500 DECLARATIVES.
013600
013700 MULTIPLE-SECTION SECTION.
  
```

Figure 7-9 (Part 6 of 10). Example of the Use of Multiple Device Files

```

013800 USE AFTER STANDARD EXCEPTION PROCEDURE ON MULTIPLE-FILE.
013900 MULTIPLE-PARAGRAPH.
014000
135 014100 WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
014200
136 014300 MOVE "FILE NAME IS:" TO DESCRIPTION OF DETAIL-LINE.
137 014400 MOVE "MULTIPLE-FILE" TO DETAIL-VALUE OF DETAIL-LINE.
138 014500 WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
014600
139 014700 MOVE "FILE STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
140 014800 MOVE MULTIPLE-FS1 TO DETAIL-VALUE OF DETAIL-LINE.
141 014900 WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
015000
142 015100 9 MOVE "EXTENDED STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
143 015200 9 MOVE MULTIPLE-FS2 TO DETAIL-VALUE OF DETAIL-LINE.
144 015300 WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
015400
145 015500 9A ACCEPT STATION-ATTR FROM ATTR.
146 015600 9A MOVE "FILE ATTRIBUTES ARE:" TO DESCRIPTION OF DETAIL-LINE.
147 015700 9A MOVE STATION-ATTR TO DETAIL-VALUE OF DETAIL-LINE.
148 015800 WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
149 015900 STOP RUN.
016000
016100 TERMINAL-SECTION SECTION.
016200 USE AFTER STANDARD EXCEPTION PROCEDURE ON TERMINAL-FILE.
016300 TERMINAL-PARAGRAPH.
016400
150 016500 WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
016600
151 016700 MOVE "FILE NAME IS:" TO DESCRIPTION OF DETAIL-LINE.
152 016800 MOVE "TERMINAL-FILE" TO DETAIL-VALUE OF DETAIL-LINE.
153 016900 WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
017000
154 017100 MOVE "FILE STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
155 017200 MOVE TERMINAL-FS1 TO DETAIL-VALUE OF DETAIL-LINE.
156 017300 WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
017400
157 017500 STOP RUN.
017600
017700 PASSWORD-SECTION SECTION.
017800 USE AFTER STANDARD EXCEPTION PROCEDURE ON PASSWORD-FILE.
017900 PASSWORD-PARAGRAPH.
018000
158 018100 WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
018200
159 018300 MOVE "FILE NAME IS:" TO DESCRIPTION OF DETAIL-LINE.
160 018400 MOVE "PASSWORD-FILE" TO DETAIL-VALUE OF DETAIL-LINE.
161 018500 WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
018600
162 018700 MOVE "FILE STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
163 018800 MOVE PASSWORD-FS1 TO DETAIL-VALUE OF DETAIL-LINE.
164 018900 WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
019000
165 019100 STOP RUN.
019200

```

Figure 7-9 (Part 7 of 10). Example of the Use of Multiple Device Files

```

5714CB1 R05 M00 830610          COBOL SOURCE LISTING          SAMPLEMF          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN 5 COPYNAME

019300 END DECLARATIVES.
019400
019500 *****
019600 MAIN PROGRAM LOGIC BEGINS HERE
019700 *****
019800
019900 MAIN-LINE SECTION.
020000 MAIN-LINE-PARAGRAPH.
020100
166 020200 OPEN I-D MULTIPLE-FILE.
020300 INPUT TERMINAL-FILE.
020400 I-N PASSWORD-FILE.
020500 OUTPUT PRINTER-FILE.
020600
167 020700 MOVE 1 TO COUNTER.
168 020800 SET NOT-END-OF-TERMINAL-LIST TO TRUE.
020900 PERFORM
169 021000 FILL-TERMINAL-LIST UNTIL END-OF-TERMINAL-LIST.
021100
021200 PERFORM
170 021300 ACQUIRE-AND-INVITE-TERMINALS VARYING COUNTER FROM 1 BY 1
021400 UNTIL COUNTER GREATER THAN NO-OF-TERMINALS.
021500
171 021600 MOVE 1 TO COUNTER.
172 021700 SET DATA-AVAILABLE TO TRUE.
021800 PERFORM
173 021900 POLL-TERMINALS UNTIL NO-DATA-AVAILABLE.
022000
022100 PERFORM
174 022200 DROP-TERMINALS VARYING COUNTER FROM 1 BY 1
022300 UNTIL COUNTER GREATER THAN NO-OF-TERMINALS.
022400
175 022500 CLOSE MULTIPLE-FILE.
022600 TERMINAL-FILE.
022700 PASSWORD-FILE.
022800 PRINTER-FILE.
022900
176 023000 STOP RUN.
023100
023200 *****
023300 PROCEDURES
023400 *****
023500
023600 PROCEDURE-SECTION SECTION.
023700 FILL-TERMINAL-LIST.
023800
177 023900 READ TERMINAL-FILE RECORD INTO LIST-OF-TERMINALS(COUNTER)
024000 AT END
178 024100 SET END-OF-TERMINAL-LIST TO TRUE
179 024200 SUBTRACT 1 FROM COUNTER
180 024300 MOVE COUNTER TO NO-OF-TERMINALS.
024400
181 024500 ADD 1 TO COUNTER.
024600
024700 ACQUIRE-AND-INVITE-TERMINALS.

```

Figure 7-9 (Part 8 of 10). Example of the Use of Multiple Device Files

```

5714CB1 R05 M00 830610          COBOL SOURCE LISTING          SAMPLEMF          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN 5 COPYNAME

024800
182 024900 ACQUIRE LIST-OF-TERMINALS(COUNTER) FOR MULTIPLE-FILE.
183 025000 WRITE MULTIPLE-REC
025100 FORMAT IS "SIGNON"
025200 TERMINAL IS LIST-OF-TERMINALS(COUNTER).
025300
025400
025500 POLL-TERMINALS.
025600
184 025700 READ MULTIPLE-FILE RECORD.
025800
185 025900 IF MULTIPLE-FS2 = "0310" THEN
186 026000 SET NO-DATA-AVAILABLE TO TRUE.
026100
187 026200 IF DATA-AVAILABLE THEN
188 026300 MOVE MULTIPLE-DEVICE-NAME TO CURRENT-TERMINAL
189 026400 PERFORM PASSWORD-VALIDATION.
026500
026600
026700 PASSWORD-VALIDATION.
026800
190 026900 MOVE CURRENT-TERMINAL TO PASSKEY OF PASSWORD-REC.
191 027000 READ PASSWORD-FILE RECORD.
027100
192 027200 IF PASSWORD OF SIGNON-I = PASSWORD OF PASSWORD-REC THEN
193 027300 CALL "UPDATE" USING CURRENT-TERMINAL
ELSE
194 027400 MOVE "INVALID PASSWORD" TO WRONG OF SIGNON-U.
027500
195 027600
027700 WRITE MULTIPLE-REC FORMAT IS "SIGNON"
027800 TERMINAL IS CURRENT-TERMINAL.
027900
028000
028100 DROP-TERMINALS.
028200
196 028300 DROP LIST-OF-TERMINALS(COUNTER) FROM MULTIPLE-FILE.
028400
***** END OF SOURCE *****

```

Figure 7-9 (Part 9 of 10). Example of the Use of Multiple Device Files

- 1** ATTR is the mnemonic-name associated with the function-name ATTRIBUTE-DATA. ATTR will be used in the ACCEPT statement to obtain attribute data for the TRANSACTION file MULTIPLE-FILE. See item **9A** .
- 2** File MULT must have been created using the CRTMXDF CL command, where the ACQPGMDEV parameter has a value of *NONE and the MAXPGMDEVE parameter has a value greater than 1. The WAITRCD parameter specifies the wait-time for READ operations on the file. The WAITRCD parameter must have a value greater than 0.
- 3** MULTIPLE-FS2 is the extended file status for the TRANSACTION file MULTIPLE-FILE. This variable has been declared in the WORKING-STORAGE section of the program. See item **7** .
- 4** MULTIPLE-CONTROL-AREA is the control area for the TRANSACTION file MULTIPLE-FILE. This variable will be used to determine which program device was signed on to. See item **15** .
- 5** The data description for MULTIPLE-REC has been defined using the COPY DDS statement. Note that only the fields which are copied are named fields. Refer to the DDS of this example for comments regarding the DDS used.
- 6** Format SIGNON is the format with the INVITE keyword. This is the format that will be used to invite devices via the WRITE statement.
- 7** This is the declaration for the extended file-status MULTIPLE-FS2. It is a 4-byte field which is subdivided into a major return code (first two bytes) and a minor return code (last two bytes).
- 8** STATION-ATTR is the structure which will be used by the ACCEPT statement to hold the attribute data for the TRANSACTION file MULTIPLE-FILE. See item **9A** .
- 9** In this statement the extended file status MULTIPLE-FS2 is being written.
- 9A** This is an example of accepting attribute-data for the TRANSACTION file MULTIPLE-FILE. Because we are not interested in a specific program device, but rather the last program device used, the FOR phrases are not used with the ACCEPT.
- 10** This statement opens the TRANSACTION file MULTIPLE-FILE. Since the ACQPGMDEV parameter of the CRTMXDF command has the value *NONE, no program devices are implicitly acquired during this open.
- 11** This statement acquires the program device contained in the variable LIST-OF-TERMINALS (COUNTER), for the TRANSACTION file MULTIPLE-FILE.
- 12** This WRITE statement is inviting the program device specified in the TERMINAL phrase. We know it is inviting the program device because the format SIGNON has the DDS keyword INVITE associated with it. Refer to item **13** .
- 13** This READ statement will read from any invited program device. See item **12** . If the wait time expires before anyone inputs to the invited devices, the extended file status will be set to "0310" and execution will continue. See item **14** .
- 14** In this statement, the extended file status for MULTIPLE-FILE is being checked to see if the wait-time expired.
- 15** The program device name stored in the control area is used to determine which program device was signed on to. See item **4** .
- 16** This DROP statement detaches the program device contained in the variable LIST-OF-TERMINALS from the TRANSACTION file MULTIPLE-FILE.

Figure 7-9 (Part 10 of 10). Example of the Use of Multiple Device Files

Program Described Transaction Files

Normally, COBOL TRANSACTION files are externally described. However, if these files are program described, only simple display formatting can be performed. All field level descriptions are defined in the COBOL program.

Packed or binary data (COMP, COMP-3, or COMP-4) should not be sent to a display station as output data. Such data can contain display station control characters which can cause unpredictable results.

See the *CPF Programmer's Guide* for more information about using program described display files.

Environment Division

File-Control Entry

The TRANSACTION file must be named by a file-control entry in the FILE-CONTROL paragraph. This entry also specifies other information related to the file.

Format

```
SELECT file-name  
  
ASSIGN TO assignment-name-1 [ , assignment-name-2 ] ...  
  
ORGANIZATION IS TRANSACTION  
  
[ ACCESS MODE IS { SEQUENTIAL  
                  DYNAMIC, RELATIVE KEY IS data-name-3 } ]  
[ FILE STATUS IS data-name-1 [ , data-name-5 ] ]  
[ CONTROL-AREA IS data-name-6 ] .
```

ASSIGN Clause

The ASSIGN clause associates the TRANSACTION file with a display file, communications file, BSC file, or mixed file through the use of assignment-name-1.

Assignment-name-1 has the following structure:

device [- system/38 file name [- attribute]]

Device specifies the type of device associated with the file. The value must be WORKSTATION.

System/38 file name is a 1- to 10-character external name of the display file, communications file, BSC file, or mixed file specified on the create device file commands, CRTDSPF, CRTCMNF, CRTBSCF, or CRTMXDF.

Attribute specifies the file level option for a separate indicator area, SI. See "INDICATORS" earlier in this chapter.

The second and subsequent assignment-names are syntax-checked, but are treated as documentation.

ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of a file. TRANSACTION organization signifies interaction between a COBOL program and either a work station user or another system.

TRANSACTION Organization: TRANSACTION processing can be characterized as the random arrival of a record from one of multiple possible sources followed by appropriate processing, and finally, by the output of results or feedback information of some type to the source of the record.

In some cases, all records are homogenous; that is, a logical transaction is completed with one exchange of records. In other situations, a series of records is passed back and forth in a logical progression with various record types either being selected by the initiator or as part of the processing based on input data values.

Each transaction can be processed by a different program, or multiple transactions can be processed by the same program, depending on the system environment.

The initiation of a transaction can cause a program to be scheduled to process the transaction.

A transaction can consist of a series of alternating requests and responses (a dialogue). Each request and response can consist of multiple logical records.

ACCESS MODE Clause

The ACCESS MODE clause and the RELATIVE KEY clause are discussed under “FILE-CONTROL Paragraph” in Chapter 3.

For files with TRANSACTION organization, the access mode can be SEQUENTIAL or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, the format name contained in the format name field of the control area specifies which record was accessed. When ACCESS IS SEQUENTIAL is specified for a TRANSACTION file, the RELATIVE KEY data item must not be specified.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request. Random accessing of a TRANSACTION file is only valid if subfile processing is being performed. For subfile processing, ACCESS IS DYNAMIC *must* be specified.

RELATIVE KEY Clause

The RELATIVE KEY clause specifies the relative record number for a specific record in a subfile. The RELATIVE KEY data item, data-name-3, must be defined as an unsigned integer and must not be defined in a record description entry associated with the TRANSACTION file.

FILE STATUS Clause

Data-name-1 and general considerations about the FILE STATUS clause are described under “FILE-CONTROL Paragraph” in Chapter 3.

Data-name-5 identifies the extended file status data item, which contains major and minor return codes. These major and minor return codes can, in some cases, indicate I-O errors when the file status code does not.

Data-name-5 must be defined in the Data Division as a 4-byte alphanumeric data item, and must *not* be defined in the File Section. The first two bytes of the extended file status data item contain the major return code, and the second two bytes contain the minor return code. Return codes are moved into data-name-5 after any input or output operation (except the ACCEPT or CLOSE statement) on the TRANSACTION file. The values placed in data-name-5 can also be accessed by the ACCEPT statement using the I-O-FEEDBACK function-name. For more information about the major and minor return codes see the *CPF Programmer's Guide* and the *Data Communications Programmer's Guide*.

CONTROL-AREA Clause

The CONTROL-AREA clause specifies device dependent and system dependent information that is used to control input/output operations for TRANSACTION files.

Data-name-6 is a CONTROL-AREA data item that must be defined in the LINKAGE SECTION or WORKING-STORAGE SECTION. Data-name-6 is assumed to have the following format:

```
01 data-name-6.
  02 data-name-12 PIC X(2).
      (Function key feedback field)
  02 data-name-11 PIC X(10)
      (Program device name)
  02 data-name-10 PIC X(10)
      (Record format)
```

Data-name-6 must be 2, 12, or 22 characters long. Based upon the length of data-name-6, the compiler assumes the availability of key feedback bytes, the program device name, and record format.

Programming Note: For a mixed file, the actual name of a device may be different than the program device name (data-name-11).

Information is moved into data-name-6 for each READ operation from a file that has been assigned to a WORKSTATION device type. The information is valid only if the READ operation is successfully completed (provided the wait time has not expired). The information is in the fixed format as shown in the following example:

```
FILE-CONTROL.
SELECT SCREEN-FILE
  ASSIGN TO WORKSTATION-MYFMTS
  ORGANIZATION IS TRANSACTION
  CONTROL-AREA IS
  TRANSACTION-CONTROL-AREA.
.
.
WORKING-STORAGE SECTION.
01 TRANSACTION-CONTROL-AREA.
* FEEDBACK ITEM
  02 COMMON-AREA.
    03 FUNCTION-KEY PIC XX.
    03 TERMINAL-ID PIC X(10).
  02 FORMAT-NAME PIC X(10).
```

Each field in the TRANSACTION-CONTROL-AREA data item in the example is described as follows:

- **FUNCTION-KEY:** A two-digit number inserted in the field by the work station interface that identifies which function key the operator pressed to initiate the transaction. The codes are as follows:

00	Enter key
01-24	Command keys 1 through 24
90	Roll Up key
91	Roll Down key
92	Print key
93	Help key
94	Clear key
95	Home key
99	Undefined

Any function keys for which feedback information is desired must be defined for the display file or mixed file using DDS. The Print key must also be optioned by a response indicator before feedback information can be provided in the function key field of the CONTROL-AREA data-name.

- **TERMINAL-ID:** The *program device name*.
- **FORMAT-NAME:** The DDS record format name that was referenced by the last I-O statement executed.

Data Division

File Description Entry

A file description entry consists of a level indicator (FD), a file-name, and a series of independent clauses. For a TRANSACTION file, the independent clauses allowed are the RECORD CONTAINS clause the LABEL RECORDS clause, and the DATA RECORDS clause. Only the LABEL RECORDS clause is required.

Format

```
[ FD file-name
  [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]
  [ LABEL { RECORD IS } { OMITTED }
    { RECORDS ARE } { STANDARD } ]
  [ DATA { RECORD IS }
    { RECORDS ARE } data-name-3 [ , data-name-4 ] . . . ] .
  { record-description-entry } . . . ]
```

The LABEL RECORDS clause specifies whether or not labels are present. This clause is required in every file description entry. This clause is syntax-checked, but is treated as documentation.

The RECORD CONTAINS clause and the DATA RECORDS clause are described in Chapter 4.

Boolean Data Facilities

The use of Boolean data and the use of indicators is described under “INDICATORS” earlier in this chapter.

Procedure Division

See Appendix I for a file structure support summary.

ACCEPT Statement

The ACCEPT statement retrieves information (attribute data) about a particular program device associated with a TRANSACTION file.

Format

```
ACCEPT identifier-1 FROM mnemonic-name  
[FOR { identifier-2 }  
  literal] [FOR file-name] ]
```

This format of the ACCEPT statement may only be used for files with an organization of TRANSACTION. If the file is not open at the time the ACCEPT is executed, message CBE7205 is issued and execution terminates. Mnemonic-name must be associated with the function-name ATTRIBUTE-DATA in the SPECIAL-NAMES paragraph.

If file-name is not specified, the default file for the ACCEPT statement is the first TRANSACTION file specified in a SELECT clause of the FILE-CONTROL paragraph.

Literal or the contents of identifier-2, if specified, indicates the program device name for which attribute data is made available. This device must have been defined (through a CRTxxxF, CHGxxxF, OVRxxxF, or ADDxxxDEVE CL command, where xxx has a value of BSC, CMN, or DSP) as available to be acquired by the file, but need not have actually been acquired. Literal, if specified, must be non-numeric and 10 characters or less in length. The contents of identifier-2, if specified, must be an alphanumeric data item 10 characters or less in length. If an invalid program device name is specified, message CBE7205 is issued and execution terminates.

If both FOR phrases are omitted (indicating the default TRANSACTION file is being used) the ACCEPT statement uses the program device from which a

READ, WRITE, REWRITE, or ACCEPT (Attribute Data) operation on the default file was most recently performed. If the only prior operation on the file was an OPEN, the ACCEPT statement uses the program device implicitly acquired by the file when the file was opened. When both FOR phrases are omitted, a program device must have been acquired in order to use this format of the ACCEPT statement.

Program device attributes are moved into identifier-1 from the appropriate attribute data format, according to the rules for a group MOVE without the CORRESPONDING phrase.

Attribute Data Formats

The attribute data retrieved by the ACCEPT statement has two different formats, depending on whether the data is for a work station or for a communications device. See Appendix I for format descriptions.

Programming Notes: For a mixed file containing different types of program devices, you may need to test the attribute data to determine the type of program device for which information has been returned. This is done by testing the first byte of the attribute data, which has different values for work stations and communications devices.

The ATTRIBUTE-DATA mnemonic name can be used *only* to obtain information about a program device acquired by a TRANSACTION file. Attribute data does *not* provide information about the status of a completed or attempted I-O operation. To obtain information about I-O operations, use the Format 3 ACCEPT statement with the I-O-FEEDBACK or OPEN-FEEDBACK mnemonic names. For more information about these mnemonic names, see "ACCEPT Statement" in Chapter 5.

ACQUIRE Statement

The ACQUIRE statement acquires a program device for a TRANSACTION file.

Format

<u>ACQUIRE</u> { identifier literal } <u>FOR</u> file-name

Literal or the contents of identifier indicates the program device name to be acquired by the specified file. Literal, if specified, must be non-numeric and 10 characters or less in length. Identifier, if specified, must refer to an alphanumeric data item 10 characters or less in length.

File-name must be the name of a file with an organization of TRANSACTION, and the file must be open when the ACQUIRE statement is executed. A compilation error message is issued if the organization is not TRANSACTION.

For a description of conditions that must be met before a device can be acquired, see the *CPF Programmer's Guide*.

Successful completion of the ACQUIRE operation makes the program device available for input and output operations. If the ACQUIRE is unsuccessful, the file status value is set to 9H and any applicable USE AFTER EXCEPTION/ERROR procedure is invoked.

Only one program device may be implicitly acquired when a file is opened. If a file is a mixed file, the single implicitly acquired program device is determined by the ACQPGMDEV parameter of the CRTMXDF CL command. If the file is a display file, the single implicitly acquired program device is determined by the first entry in the DEV parameter of the CRTDSPF, CHGDSPF, or OVRDSPF CL command. Additional program devices *must* be explicitly acquired. If the file is a communications or BSC file, the single implicitly acquired device is determined by the DEV parameter of the CRTCMNF, CRTBSCF, CHGCMNF, CHGBSCF, OVRCMNF, or OVRBSCF CL command. Communications and BSC files can never acquire multiple program devices.

A program device is explicitly acquired by using the ACQUIRE statement. For a mixed file, that device must have been added to the file with an ADDDSPDEVE, ADDCMNDEVE or ADDBSCDEVE CL command before the file was opened. For a display file, the program device must have been specified in the DEV parameter when the file was created, changed, or overridden, and before the OPEN is issued for the file.

For more information about these commands, see the *CL Reference Manual* and the *CPF Programmer's Guide*.

The ACQUIRE statement can also be used as an aid in recovering from I-O errors. For more information, see "Communications Recovery" in Chapter 9.

CLOSE Statement

The CLOSE statement terminates the processing of files.

Format

```
CLOSE file-name-1 [WITH LOCK ]  
      [file-name-2 [WITH LOCK ]]. . .
```

For a further discussion of the CLOSE statement, see "CLOSE Statement" in Chapter 5.

DROP Statement

The DROP statement releases a program device that has been acquired by a TRANSACTION file.

Format

```
DROP { identifier } FROM file-name  
      { literal }
```

Literal or the contents of identifier indicates the program device name of the device to be dropped. Literal, if specified, must be non-numeric and 10 characters or less in length. Identifier, if specified, must refer to an alphanumeric data item, 10 characters or less in length.

File-name must refer to a file with an organization of TRANSACTION, and the file must be open in order to be used in the DROP statement. If no DROP statement is issued, program devices attached to a TRANSACTION file are implicitly released when that file is finally closed.

Program devices specified in a DROP statement must have been acquired by the TRANSACTION file, either through an explicit ACQUIRE or through an implicit ACQUIRE at OPEN time.

After successful execution of the DROP statement, the program device is no longer available for input or output operations through the TRANSACTION file. The device may be reacquired if necessary. The contents of the record area associated with a released program device are no longer available, even if the device is reacquired.

If the DROP statement is unsuccessful, any applicable USE AFTER EXCEPTION/ERROR procedures are executed.

The DROP statement can also be used as an aid in recovering from I-O errors. For more information, see "Communications Recovery" in Chapter 9.

OPEN Statement

The OPEN statement initiates the processing of files.

Format

```
OPEN I-O file-name-1 [ file-name-2 ] . . .
```


A TRANSACTION file must be opened in the I-O mode. For a further discussion of the OPEN statement, see “OPEN Statement” in Chapter 5.

The OPEN statement can cause a program device to be implicitly acquired for a TRANSACTION file. For a further discussion about the acquiring of program devices, see “ACQUIRE Statement” earlier in this chapter.

Common Processing Facilities

The following discussion on FORMAT, INDICATORS, SUBFILE, and TERMINAL phrases relates to the READ, REWRITE, and WRITE statements.

FORMAT Phrase

The literal or identifier specified must be a character-string of 10 characters or less in length.

Multiple data records, each with a different format, can be concurrently active for a TRANSACTION file. If the FORMAT phrase is specified, it must specify a valid format name that is defined to the system, and the I-O operation must be performed on a data record of the same format. If the format is an invalid name or if it does not exist, the FILE STATUS data item, if specified, is set to a value of 9K and the contents of the record area are undefined.

DB-FORMAT-NAME Special Register

After the execution of an input/output statement for a TRANSACTION file, the DB-FORMAT-NAME special register is modified according to the following rules:

- If the input/output operation is successful, the record format name is implicitly moved to the special register after completion of the input/output operation.
- If the input/output operation is unsuccessful, DB-FORMAT-NAME contains the record format name used in the last successful input/output operation.

When the FORMAT phrase is not specified, DB-FORMAT-NAME can be used if the file contains a default record format name. The default value is always moved to the DB-FORMAT-NAME special register.

DB-FORMAT-NAME is implicitly defined as PICTURE X(10).

INDICATORS Phrase

The identifier specified in the INDICATORS phrase must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has elementary Boolean data items subordinate to it.

When a data record is written or rewritten, indicators can be written or rewritten with it. The indicators can control how the record is displayed and also the various Data Management functions.

When a data record is read, indicators can be read with it. The indicators can be used to pass information about the data record and how it was entered into the user program.

The user determines, when he defines a format using DDS, what functions are to be controlled by indicators, and which indicator(s) controls a particular function.

See "INDICATORS" earlier in this chapter for more information on the INDICATORS phrase.

SUBFILE Phrase

When the SUBFILE phrase is specified, it indicates that all formats referenced by the statement are subfiles. When SUBFILE is not specified in a TRANSACTION I-O statement, it indicates that none of the formats referenced by the statement are subfiles. This information is not verified at compile-time. If it is specified incorrectly, an incorrect program is generated; when the program is executed, the FILE STATUS data item, if specified, is set to a value of 92 (logic error), and the contents of the record are undefined.

When SUBFILE is not specified, the RELATIVE KEY data item associated with the file, if specified, is not referenced or changed by the I-O operation.

When SUBFILE is specified, a RELATIVE KEY data item must be defined for the file. Its value is referenced, and sometimes changed, by the I-O operation. See each I-O statement associated with SUBFILE operations for a detailed description of when and how the RELATIVE KEY data item is changed.

The SUBFILE phrase can be specified only for display files, and for display devices in a mixed file.

TERMINAL Phrase

When the TERMINAL phrase is specified, it indicates a specific program device is to be used for a READ, WRITE, or REWRITE operation on a TRANSACTION file.

The TERMINAL phrase can be omitted for I-O operations on single device files, since the single device is always used.

If the **TERMINAL** phrase is omitted for an I-O operation on a **TRANSACTION** file that has acquired multiple program devices, the program device that last attempted a **READ**, **WRITE**, **REWRITE**, **ACQUIRE**, **DROP**, or **ACCEPT (Attribute Data)** operation on the file is used. If the only prior operation on the file was an **OPEN**, the default program device used is the program device implicitly acquired by the **TRANSACTION** file when the file was opened. A run-time error message occurs if no program device is acquired when the file is opened.

For a **READ** statement with both the **TERMINAL** phrase and the **NO DATA** phrase specified, the imperative-statement in the **NO DATA** phrase is executed only if data is not immediately available from the program device specified by the **TERMINAL** phrase.

If the **TERMINAL** phrase is specified and the data-item or literal has a value of blanks, the phrase is treated at execution time as if it were not specified.

READ Statement

The **READ** statement makes available a record from a device, using a named format. If the format is a subfile, the **READ** statement makes available a specified record from that subfile.

Format 1

Format 1 – Nonsubfile Format

```
READ file-name RECORD  
  
[ INTO identifier-1 ]  
  
[ FORMAT IS { identifier-2 }  
             { literal-1 } ]  
  
[ TERMINAL IS { identifier-3 }  
              { literal-3 } ]  
  
[ { INDICATOR  
  { INDICATORS [ IS ARE ] } identifier-4  
  INDIC } ]  
  
[ NO DATA imperative-statement-1 ]  
  
[ AT END imperative-statement-2 ]
```

Format 1 is used only to read a format that is not a subfile. The **RELATIVE KEY** data item, if specified in the **FILE-CONTROL** entry, is not used. The Format 1 **READ** statement is not valid for a subfile record. However, a Format 1 **READ** statement for the subfile control record format must be used

to place those subfile records that were updated on a display into the subfile.

If the data is available, it is returned in the record area. The names of the record format and the program device are returned in the I-O-FEEDBACK area in the CONTROL-AREA.

The READ statement is valid only when there are acquired devices for the file. If a READ is executed and there are no acquired devices, the file status is set to 92 (logic error).

The manner in which the Format 1 READ statement functions depends on:

- If the READ is for a single device file or a multiple device file
- If a specific program device has been requested through the TERMINAL phrase
- If a specific record format has been requested through the FORMAT phrase
- If the NO DATA phrase has been specified.

In the following discussions, references to “data available or returned” include the situation where only the response indicators are set. This is so even when a separate indicator area is used and the indicators are not returned in the record area for the file.

The following chart shows the possible combinations of phrases, and the function performed for a single device file or a multiple device file. For example, if TERMINAL is N, FORMAT is N, and NO DATA is N, then the single device is D and multiple device is A.

	Phrase	Y = Yes	N = No
Checked at Compilation	TERMINAL ¹	N N N N	Y Y Y Y
	FORMAT ¹	N N Y Y	N N Y Y
	NO DATA	N Y N Y	N Y N Y
Determined at Execution	Single Device	D C D B	D C D B
	Multiple Device	A A D B	D C D B

Codes A through D are explained below.

Code A – *Read From Invited Program Device (Multiple Device Files only)*

This type of READ receives data from the first invited program device that has data available. An invited program device is a work station or communications device (LU1, BSC, or PEER) that has been invited to send input. Inviting is done by writing to the program device with a format that has the DDS keyword INVITE specified. Once an invited program device is actually read from, it is no longer invited. That program device will not be

¹ If the phrase is specified and the data item or literal is blank, the phrase is treated at execution time as if it were not specified.

used for input by another READ statement unless reinvited, or unless a READ is directed to it specifying the TERMINAL phrase or FORMAT phrase.

The record format returned from the program device is determined by the system. See the chapter on display devices in the *CPF Programmer's Guide* for information on how this is determined for work stations. For communications devices, see the *CL Reference Manual* for information on the FMTSLT parameter on the ADDBSCDEVE and ADDCMNDEVE commands.

This READ can complete without returning any data in the following cases:

1. There are no invited devices. This is the AT END condition, which occurs when:
 - There are no invited devices.
 - For a PEER device, another READ is done after a detach signal is received.
 - For an LU1 device, the session is terminated by the host.
2. A controlled cancel of the job occurs. This results in a file status value of 9A and a major-minor return code value of 0309.
3. The NO DATA phrase is omitted and the specified wait time expires. This results in a file status value of 00 and a major-minor return code value of 0310. The specified wait time is the value entered on the WAITRCD parameter for the file.
4. The NO DATA phrase is specified and there is no data immediately available when the READ is executed.

If data is available, it is returned in the record area. The record format is returned in the I-O-FEEDBACK area and in the CONTROL-AREA. For more information about "reading from invited program devices," see the *CPF Programmer's Guide*.

Code B – Read From One Program Device (Invalid combination)

A compilation time message is issued and the NO DATA phrase is ignored. See the table entry for the same combination of phrases with the NO DATA phrase omitted.

Code C – Read From One Program Device (with NO DATA phrase)

This function of the READ statement never causes program execution to stop and wait until data is available. Either the data is immediately available or the NO DATA imperative-statement is executed.

This READ function can be used to periodically check if data is available from a particular program device (either the default program device or one specified by the TERMINAL phrase). This checking for data is done in the following manner:

1. The program device is determined as follows:
 - a. If the TERMINAL phrase was omitted or contains blanks, the default program device is used. The default program device is the one used by the last attempted READ, WRITE, REWRITE, ACQUIRE, or DROP statement. If none of the above I-O operations were previously executed, the default program device is the first program device acquired.
 - b. If the TERMINAL phrase was specified, the indicated program device is used.
2. A check is done to determine if data is available and if the program device is invited.
3. If data is available, that data is returned in the record area and the program device is no longer invited. If no data is immediately available, the NO DATA imperative-statement is executed and the program device remains invited.
4. If the program device is not invited, the AT END condition exists and the file status is set to 10.

Code D – *Read From One Program Device (without NO DATA Phrase)*

This READ always waits for data to be made available. Even if the job receives a controlled cancel, or a WAITRCD time is specified for the file, the program will never regain control from the READ statement. This READ operation is performed in the following manner:

1. The program device is determined as follows:
 - a. If the TERMINAL phrase is omitted or contains a blank value, the default program device is used. The default program device is the program device used by the last attempted READ, WRITE, REWRITE, ACQUIRE, DROP or ACCEPT (Attribute Data) statement. If none of these operations has been done, the program device implicitly acquired when the file was opened is used. If there are no acquired devices, the AT END condition exists.
 - b. If the TERMINAL phrase is specified, the indicated program device is used.
2. The record format is determined as follows:
 - a. If the FORMAT phrase is omitted or contains blanks, the record format returned is determined by the system. For information on how the record format is determined for work station devices, see the chapter on “Display Device Support” in the *CPF Programmer’s Guide*. For information about how the record format is determined for communications devices, see the section on the FMTSLT parameter on such commands as the CRTBSCF, CRTCMNF, ADDBSCDEVE, and ADDCMNDEVE commands in the *CL Reference Manual*.

- b. If the **FORMAT** phrase is specified, the indicated record format is returned. If the data available does not match the requested record format, a file status of **9K** is set.
3. Program execution stops until data becomes available. The data is returned in the record area after the **READ** statement is executed. If the program device was previously invited, it will no longer be invited after this **READ** statement.
4. The **AT END** condition can only occur for **LU1** or **PEER** devices. When there are multiple **LU1** or **PEER** devices acquired for a **TRANSACTION** file, each individual device can cause an **AT END** condition to occur. The **AT END** condition occurs when:
 - For a **PEER** device, another **READ** is done after a detach signal is received. Since a detach signal can be sent with or without data, check the major-minor return codes to determine if there was any data.
 - For an **LU1** device, the session is terminated by the host.

AT END Phrase

Imperative-statement-2 is executed when the **AT END** condition is detected.

FORMAT Phrase

Literal-1 or identifier-2 specifies the name of the record format to be read. Literal-1, if specified, must be nonnumeric and 10 characters or less in length. Identifier-2, if specified, must refer to an alphanumeric data item, 10 characters or less in length. If identifier-2 contains blanks, the **READ** statement is executed as if the **FORMAT** phrase was omitted.

NO DATA Phrase

When the **NO DATA** phrase is specified, the **READ** statement will determine whether data is immediately available. If data is available, the data is returned in the record area. If no data is immediately available, imperative-statement-1 is executed. The **NO DATA** phrase prevents the **READ** statement from waiting for data to become available.

TERMINAL Phrase

Literal-2 or identifier-3 specifies the program device name. Literal-2, if specified, must be nonnumeric and 10 characters or less in length. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or less in length. The program device must have been acquired before the **READ** statement is executed. If identifier-3 contains blanks, the **READ** statement is executed as if the **TERMINAL** phrase was omitted. For a single device file, the **TERMINAL** phrase can be omitted. The program device is assumed to be that single device.

If the **TERMINAL** phrase is omitted for a **READ** of a **TRANSACTION** file that has acquired multiple program devices, the default program device is used. See the general discussion about the **TERMINAL** phrase in the “Common Processing Facilities” section earlier in this chapter for details about how the default program device is determined.

Format 2

Format 2 – Subfile Format

```

READ SUBFILE file-name

[ NEXT MODIFIED ] RECORD

[ INTO identifier-1 ]

[ FORMAT IS { identifier-2 }
  { literal-1 } ]

[ TERMINAL IS { identifier-3 }
  { literal-2 } ]

[ { INDICATOR
  { INDICATORS [ IS ]
  { INDIC [ ARE ] } } identifier-4 ]

[ INVALID KEY imperative-statement-1 ]

[ AT END imperative-statement-2 ]

```

Format 2 is used only to read a format that is a subfile. The **AT END** phrase can only be used when the **NEXT MODIFIED** phrase is specified. The **INVALID KEY** phrase must not be used when the **NEXT MODIFIED** phrase is specified.

Format 2 cannot be used for communications devices. If the subfile format of the **READ** statement is used for a communications device, the **READ** fails and a file status of 90 is set.

Random Access of Subfile Records: The **NEXT MODIFIED** phrase must not be used to randomly access records in a subfile. The **INVALID KEY** phrase can only be used for random access of subfile records.

Sequential Access of Subfile Records: The **NEXT MODIFIED** phrase must be specified to access subfile records sequentially. The **AT END** phrase can only be specified with the **NEXT MODIFIED** phrase.

NEXT MODIFIED Phrase

When **NEXT MODIFIED** is not specified, the data record made available is the record in the subfile with a relative record number that corresponds to the value of the **RELATIVE KEY** data item.

When the NEXT MODIFIED phrase is not specified, and if the RELATIVE KEY data item contains a value other than the relative record number of a record in the subfile, the INVALID KEY condition exists and the execution of the READ statement is unsuccessful.

When the NEXT MODIFIED phrase is specified, the record made available is the first record in the subfile that has been modified (has the Modified Data Tag on). For information about turning on the Modified Data Tag, see the *CPF Programmer's Guide*.

The search for the next modified record begins:

- At the beginning of the subfile if:
 - An I-O operation has been performed for the subfile control record.
 - The I-O operation cleared, initialized, or displayed the subfile.
- For all other cases, with the record following the record that was read by a previous read operation.

The value of the RELATIVE KEY data item is updated to reflect the relative record number of the record made available to the program.

If NEXT MODIFIED is specified and there is no user-modified record in the subfile with a relative record number greater than the relative record number contained in the RELATIVE KEY data item, the AT END condition exists. Imperative-statement-2, or any applicable USE AFTER ERROR/EXCEPTION procedure, if any, is then executed.

FORMAT Phrase

When a format-name is not specified, the format used is last record format written to the display device that contains input fields, input/output fields, or hidden fields. If no such format exists for the display file, the format used is the record format of the last WRITE operation to the display device.

If the FORMAT phrase is specified, literal-1 or the contents of identifier-2 must specify a format, which is active for the appropriate program device. The READ statement reads a data record of the specified format.

The FORMAT phrase should always be specified for multiple format files to ensure correct results. For more information on the FORMAT phrase, see "Procedure Division, Common Processing Facilities" in this chapter.

TERMINAL Phrase

See Format 1 above for general considerations concerning the TERMINAL phrase.

For a Format 2 READ, if the TERMINAL phrase is omitted for a file that has multiple devices acquired for it, a record is read from the subfile associated with the default program device. See the general discussion about the TERMINAL phrase in the "Common Processing Facilities" section earlier in

this chapter for more details about how the default program device is determined.

INVALID KEY Phrase

If the RELATIVE KEY data item at the time of the execution of the READ statement contains a value that does not correspond to a relative record number for the subfile, the INVALID KEY condition exists and the execution of the READ statement is unsuccessful.

For a Format 2 READ, the INVALID KEY phrase should be specified if the NEXT MODIFIED phrase is not specified and there is no applicable USE procedure specified for the file-name.

If both an INVALID KEY phrase and a USE procedure are specified for the file when the INVALID KEY condition occurs, control transfers to the INVALID KEY imperative-statement, and the USE procedure is not executed.

AT END Phrase

If NEXT MODIFIED is specified and there is no user-modified record in the subfile, the AT END condition exists, and the execution of the READ statement is unsuccessful.

The AT END phrase should be specified when the NEXT MODIFIED phrase is used, and no applicable USE procedure is specified for the file-name. If the AT END phrase and a USE procedure are both specified for a file, and the AT END condition arises, control transfers to the AT END imperative statement and the USE procedure is not executed.

For a further discussion of the READ statement (and a related topic, the INTO phrase), the INVALID KEY phrase, and the AT END phrase, see "READ Statement" in Chapter 5.

REWRITE Statement

The REWRITE statement is used to replace a subfile record that already exists in the subfile.

Format

```
REWRITE SUBFILE record-name [FROM identifier-1]

FORMAT IS { identifier-2
           literal-1 }

[ TERMINAL IS { identifier-3
               literal-2 } ]

[ { INDICATOR
  INDICATORS [ IS
              ARE ] } identifier-4 ]

[ INVALID KEY imperative-statement ]
```

The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced. A successful read operation on the record must be done prior to the rewrite operation. The record replaced in the subfile is the record in the subfile accessed by the previous read operation.

FORMAT Phrase

The record format specified in the FORMAT phrase must be the record format accessed on the previous read operation. Literal-1 or the contents of identifier-2 must be the name of the subfile format accessed on the previous READ. For more information on the FORMAT phrase, see "Procedure Division, Common Processing Facilities" earlier in this chapter.

TERMINAL Phrase

The TERMINAL phrase indicates which program device's subfile is to have a record rewritten. If the TERMINAL phrase is specified, literal-2 or identifier-3 must refer to a work station that has been acquired by the TRANSACTION file. If literal-2 or identifier-3 contains blanks, the TERMINAL phrase has no effect. The program device specified by the TERMINAL phrase must have been acquired, either explicitly or implicitly, and must have a subfile associated with the device.

Literal-2 or identifier-3 must be a valid program device name. Literal-2, if specified, must be nonnumeric and 10 characters or less. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or less in length.

If the **TERMINAL** phrase is omitted from a **TRANSACTION** file that has acquired multiple program devices, the subfile used is the subfile associated with the last program device from which a **READ** of the **TRANSACTION** file was attempted.

The **REWRITE** statement cannot be used for communications devices. If the **REWRITE** statement is used for a communications device, the operation fails and a file status of 90 is set.

INVALID KEY Phrase

If, at the time of the rewrite operation, the **RELATIVE KEY** data item contains a value that does not correspond to the relative record number of the record from the previous read operation, the **INVALID KEY** condition exists.

The **INVALID KEY** phrase should be specified for files for which an appropriate **USE** procedure is not specified. Undesirable results may occur if the **INVALID KEY** phrase is not specified, and no **USE** procedure is specified.

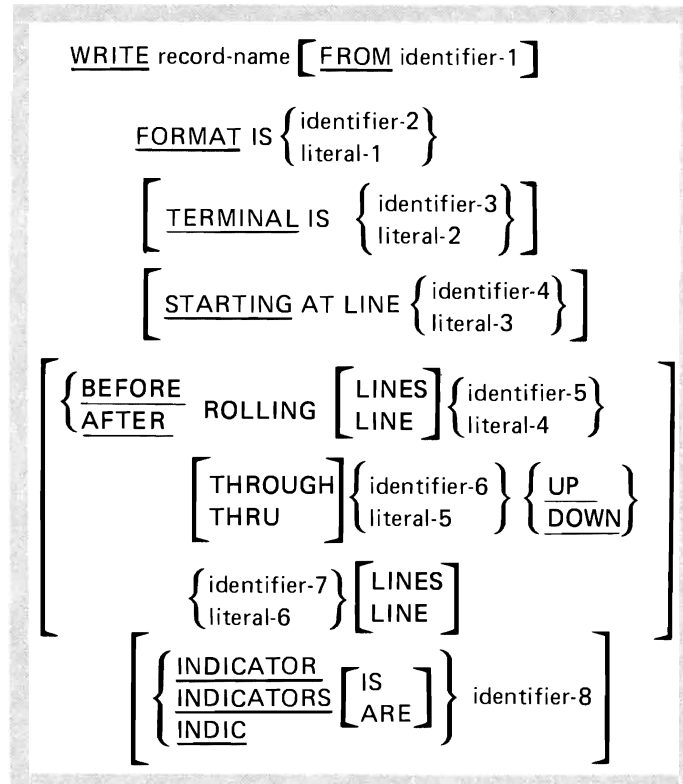
For a further discussion of the **REWRITE** statement (and the related topic, the **FROM** phrase) and the **INVALID KEY** phrase, see “**REWRITE Statement**” in Chapter 5.

WRITE Statement

The WRITE statement releases a logical record to the file.

Format 1

Format 1 – Nonsubfile Formats



TERMINAL Phrase

The TERMINAL phrase specifies the program devices to which the output record is to be sent.

The contents of literal-2 or identifier-3 must be the name of a program device previously acquired, either implicitly or explicitly, by the file. Literal-2, if specified, must be nonnumeric and 10 characters or less in length. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or less in length. A value of blanks is treated as if the TERMINAL phrase was omitted.

If only a single program device was acquired by the TRANSACTION file, the TERMINAL phrase can be omitted. That program device is always used for the WRITE.

If the TERMINAL phrase is omitted for a WRITE operation to a TRANSACTION file that has acquired multiple program devices, the default program device is used. See the general discussion about the TERMINAL phrase in the

“Common Processing Facilities” section earlier in this chapter for details about how the default program device is determined.

STARTING Phrase

The **STARTING** phrase specifies the starting line number for the record formats that use the variable start line keyword. This phrase is only valid for display devices.

The actual line number on which a field begins can be determined from the following equation:

$$\begin{array}{r} \text{Actual} \\ \text{line} \\ \text{number} \end{array} = \begin{array}{r} \text{Starting line} \\ \text{number specified} \\ \text{in the program} \end{array} + \begin{array}{r} \text{The line number} \\ \text{specified in} \\ \text{positions 39} \\ \text{through 41 of the} \\ \text{Data Description} \\ \text{Specifications form} \end{array} - 1$$

The write is successful if:

- The result of the above equation is positive and less than or equal to the number of lines on the work station screen.
- The value specified for the **STARTING** phrase is 0. In this case, a value of 1 is assumed.

The write is unsuccessful and the program terminates if:

- The result of the above equation is greater than the number of lines on the work station screen.
- The value specified for the **STARTING** phrase is negative.

If the value specified for the **STARTING** phrase is within the screen area, any fields outside of the screen area are ignored.

Literal-3 of the **STARTING** phrase must be a numeric literal. Identifier-4 must be an elementary numeric item.

To use the **STARTING** phrase, the DDS record level keyword **SLNO(*VAR)** must be specified for the format being written. If the record format does not specify this keyword, the **STARTING** phrase is ignored at execution time.

The DDS keyword **CLRL** also affects the **STARTING** phrase. **CLRL** controls how much of the screen is cleared when the **WRITE** statement is executed.

See the *CPF Reference Manual – DDS* and the *CPF Programmer’s Guide* for further information on **SLNO(*VAR)** and **CLRL**.

ROLLING Phrase

The **ROLLING** phrase allows you to move lines displayed on the work station screen. All or some of the lines on the screen can be rolled up or down. The lines vacated by the rolled lines are cleared, and can have another

screen format written into them. This phrase is only valid for display devices.

ROLLING is specified in the WRITE statement that is writing a new format to the work station screen. You must specify whether the write is before or after the roll, the range of lines you want to roll, how many lines you want to roll these lines, and whether the roll operation is up or down.

After lines are rolled, the fields on these lines retain their DDS display attributes, for example, underlining, but lose their DDS usage attributes, for example, input-capability. Fields on lines that are written and then rolled (BEFORE ROLLING phrase) also lose their usage attributes.

If any part of a format is rolled, the entire format loses its usage attributes. If more than one format exists, only the rolled formats lose their usage attributes.

When you specify the ROLLING phrase, the following general rules apply.

- The DDS record level keyword ALWROL must be specified for every record format written in a WRITE statement containing the ROLLING phrase.
- Other DDS keywords mutually exclusive with the ALWROL keyword must not be used; see the *CPF Reference Manual – DDS*.
- Either of the DDS keywords, CLRL or OVERLAY, must be specified for a record format that is to be written and rolled to prevent the display screen from being cleared when that record format is written. See the *CPF Reference Manual – DDS*.
- All the identifiers and literals must represent positive integer values.
- The roll starting line number (identifier-5 or literal-4) must not exceed the ending line number (identifier-6 or literal-5).
- The contents of lines that are rolled outside of the window specified by the starting and ending line numbers disappear.

Figure 7-9 shows an example of rolling. An initial screen format, FMT1 is written on the work station screen. The program processes this screen format and is now ready to write the next screen format, FMT2, to the work station screen. Part of FMT1 is rolled down 2 lines before FMT2 is written to the work station screen.

Execution of the following WRITE statement causes part of FMT1 to be rolled down 2 lines, and FMT2 to be written to the work station screen:

```
WRITE SCREENREC FORMAT "FMT2"  
  AFTER ROLLING LINES 14 THROUGH 20  
  DOWN 2 LINES
```

When this WRITE statement is executed, the following steps occur:

1. The contents of lines 14 through 20 are rolled down 2 lines.

- a. The contents of lines 14 through 18 now appear on lines 16 through 20.
 - b. The contents of lines 14 and 15 are vacated and cleared.
 - c. The contents of lines 19 and 20 are rolled outside the window and disappear.
2. After the rolling operation takes place, FMT2 is written to the work station screen.
 - a. Part of FMT2 is written to the area vacated by the roll operation.
 - b. Part of FMT2 is written over the data left from FMT1.
 3. When the contents of the work station screen are returned to the program by a READ statement, only the input capable fields of FMT2 are returned.

Format 2

Format 2 – Subfile Formats

```

WRITE SUBFILE record-name [FROM identifier-1]

  FORMAT IS { identifier-2
             literal-1 }

  [ TERMINAL IS { identifier-3
                 literal-2 } ]

  [ { INDICATOR [ IS ]
      INDICATORS [ ARE ]
      INDIC      } identifier-4 ]

  [ INVALID KEY imperative-statement ]

```

Format 2 can only be used for display devices. If the subfile form of the WRITE statement is used for any other type of device, the WRITE operation fails and a file status of 90 is set.

If the format is a subfile, and SUBFILE is specified, the RELATIVE KEY clause must have been specified on the SELECT clause for the file being written. The record written to the subfile is the record in the subfile identified by the format name that has a relative record number equal to the value of the RELATIVE KEY data item. (See *CPF Reference Manual – DDS*.)

TERMINAL Phrase

See Format 1 above for general considerations concerning the TERMINAL phrase.

The TERMINAL phrase specifies which program device's subfile is to have a record written to it. If the TERMINAL phrase is specified, literal-2 or identifier-3 must refer to a work station associated with the TRANSACTION file. If literal-2 or identifier-3 contains a value of blanks, the TERMINAL phrase is treated as if it was not specified. The work station specified by the TERMINAL phrase must have been acquired, either explicitly or implicitly.

If the TERMINAL phrase is omitted, the subfile used is the subfile associated with the default program device. See the general discussion about the TERMINAL phrase in the "Common Processing Facilities" section earlier in this chapter for details about how the default program device is determined.

The INVALID KEY condition exists if a record is already in the subfile with that record number, or if the relative record number specified is greater than the maximum allowable subfile record number. The INVALID KEY phrase should be specified in the WRITE SUBFILE statement for all files for which an appropriate USE procedure is not specified.

For a further discussion of the WRITE statement, the FROM phrase, and the INVALID KEY phrase, see "WRITE Statement" in Chapter 5. For information on the FORMAT phrase, see "Procedure Division, Common Processing Facilities" in this chapter.

DISPLAY BEFORE EXECUTION OF THE WRITE STATEMENT

```
UPDATE CUSTOMER ORDER RECORD

TO END THIS JOB, PRESS CMD KEY 7

ENTER YOUR OPERATOR NUMBER: ---
ENTER CUSTOMER NUMBER: -----
PRESS CMD KEY 3 TO DISPLAY OPTION MENU
```

Line 3

Line 8

Line 13

Line 14

Line 15

Line 17

Line 20

These 7 lines of FMT1 will be rolled down 2 lines.

DISPLAY AFTER EXECUTION OF THE WRITE STATEMENT

```
UPDATE CUSTOMER ORDER RECORD

TO END THIS JOB, PRESS CMD KEY 7

ITEM NUMBER ORDERED: _____
QUANTITY ORDERED: _____

ENTER CUSTOMER NUMBER: XXXXX
PRESS CMD KEY 3 TO DISPLAY OPTION MENU
```

Line 3

Line 8

Line 12

Line 14

Line 17

Line 19

These 3 lines of FMT2 have been written over the previous lines.

Figure 7-10. Example of ROLLING Operation

USE Statement

The USE statement specifies procedures for input/output error handling that are in addition to the standard procedures provided by the input/output control system.

Format

```
USE AFTER STANDARD { ERROR
                     EXCEPTION }
PROCEDURE ON { file-name-1 [ , file-name-2 ] . . . }
              { I-O }
```

See “Declaratives” in Chapter 5 for a further discussion of the USE statement.

Work Station Sample Programs

This section contains sample COBOL programs that illustrate work station applications on the System/38.

Figure 7-11 shows a basic inquiry program that uses the COBOL TRANSACTION file. The associated DDS for the files is also shown.

File		Keying Instruction	Graphic					Description	Page	of
Programmer	Date		Key							

Sequence Number	Form Type	Acd/Co/Comment (A/O/?)	Conditioning					Name	Length	Reference (R)	Data Type (B A/P/S/B A/S/X/Y/N/U/M)	Location		Functions
			Indicator	Not (N)	Indicator	Not (N)	Indicator					Not (N)	Line	
1	A	X												
2	A	*												
3	A													
4	A													
5	A													
6	A													
7	A													
8	A													
9	A													
10	A													
11	A													
12	A													
13	A													
14	A													
15	A													
16	A													
17	A													
18	A													
19	A													
20	A													
21	A													
22	A													
23	A													
24	A													
25	A													
26	A													
27	A													
28	A													
29	A													
30	A													
31	A													
32	A													
33	A													
34	A													
35	A													
36	A													
37	A													
38	A													
39	A													
40	A													
41	A													
42	A													
43	A													
44	A													
45	A													
46	A													
47	A													
48	A													
49	A													
50	A													
51	A													
52	A													
53	A													
54	A													
55	A													
56	A													
57	A													
58	A													
59	A													
60	A													
61	A													
62	A													
63	A													
64	A													
65	A													
66	A													
67	A													
68	A													
69	A													
70	A													
71	A													
72	A													
73	A													
74	A													
75	A													
76	A													
77	A													
78	A													
79	A													
80	A													

*Number of sheets per pad may vary slightly.

Figure 7-11 (Part 1 of 8). Example of a TRANSACTION Inquiry Program Using a Single Display Device

The data description specifications (DDS) for the display device file (CUSMINQ) to be used by this program describe two record formats: CUSPMT and CUSFLDS.

The CUSPMT record format contains the constant 'Customer Master Inquiry', which identifies the display. It also contains the prompt 'Customer Number' and the input field (CUST) into which the work station user enters the customer number. Five underscores appear under the input field CUST on the screen where the user is to enter the customer number. The error message 'Customer number not found' is also included in this record format. This message is displayed if indicator 99 is set on by the program. In addition, this record format defines a command attention key that the user can press to end the program. When the user presses command attention key 01, indicator 15 is set on in the COBOL program. This indicator is then used to end the program.

The CUSFLDS record format contains the constants 'Name', 'Address', 'City', 'State', 'Zip Code', and 'A/R Balance', which identify the fields to be written out from the program. This record format also describes the fields that correspond to these constants. All of these fields are described as output fields (blank in position 38) because they are filled in by the program; the user does not enter any data into these fields. To enter another customer number, the user presses the Enter key in response to this record. Notice that the CUSFLDS record is to overlay the CUSPMT record. Therefore, when the CUSFLDS record is written to the screen, the CUSPMT record remains on the screen.

In addition to describing the constants, fields, and attributes for the screen, the record formats also define the line numbers and horizontal positions in which the constants and fields are to be displayed.

Note: The field attributes are defined in a physical file (CUSMSTP) used for field reference purposes, instead of in the DDS for the display file.

Figure 7-11 (Part 2 of 8). Example of a TRANSACTION Inquiry Program Using a Single Display Device

File	Keying Instruction	Graphic							
Programmer	Date	Key							

Description	Page	of
-------------	------	----

Sequence Number	Form Type 7-And/Or Comment (A/O/*) 8-Not (N)	Conditioning					Name	Length	Reference (R)	Data Type (B A/P/S/B A/S/X/Y/N/I/N)	Decimal Positions	Usage (B/O)/B/H/M)	Location		Functions	
		Indicator Not (N)	Indicator Not (N)	Indicator Not (N)	Indicator Not (N)	Indicator Not (N)							Line	Pos		
	A															
	X					PHYSICAL	CUSMSTP									
	A					R	CUSMST						45			TEXT('Customer Master Record')
	A						CUST			5						TEXT('Customer number')
	A						NAME			25						TEXT('Customer name')
	A						ADDR			20						TEXT('Customer address')
	A						CITY			20						TEXT('Customer city')
	A						STATE			2						TEXT('State')
	A						ZIP			5	00					TEXT('Zip code')
	A						SRHCOD			6						TEXT('Customer number search code')
	A						CUSTYP			1	00					TEXT('Customer type 1=GOV 2=SCH + 3=BUS 4=PVT 5=OT')
	A						ARBAL			8	02					TEXT('Accts rec balance')
	A						ORDBAL			8	02					TEXT('A/R amt in order file')
	A						LSTAMT			8	02					TEXT('Last amount paid in A/R')
	A						LSTDAT			6	00					TEXT('Last date paid in A/R')
	A						CRDLMT			8	02					TEXT('Customer credit limit')
	A						SLSYR			10	02					TEXT('Customer sales this year')
	A						SLSLYR			10	02					TEXT('Customer sales last year')
	A					K	CUST									

The data description specifications (DDS) for the data base file that is used by this program describe one record format: CUSMST. Each field in the record format is described, and the CUST field is identified as the key field for the record format.

Figure 7-11 (Part 3 of 8). Example of a TRANSACTION Inquiry Program Using a Single Display Device

```

SEGNBR*... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7

100      IDENTIFICATION DIVISION.
200      PROGRAM-ID. INQUIRY.
300      ENVIRONMENT DIVISION.
400      CONFIGURATION SECTION.
500      SOURCE-COMPUTER. IBM-S38.
600      OBJECT-COMPUTER. IBM-S38.
700      INPUT-OUTPUT SECTION.
800      FILE-CONTROL.
900          SELECT CUST-DISPLAY
1000             ASSIGN TO WORKSTATION-CUSMING.
1100             ORGANIZATION IS TRANSACTION.
1200             CONTROL-AREA IS WS-CONTROL.
1300          SELECT CUST-MASTER
1400             ASSIGN TO DATABASE-CUSMSTP.
1500             ORGANIZATION IS INDEXED.
1600             ACCESS IS RANDOM
1700             RECORD KEY IS CUST OF CUSMST.
1800             FILE STATUS IS CM-STATUS.
1900      DATA DIVISION.
2000      FILE SECTION.
2100      FD CUST-DISPLAY
2200          LABEL RECORDS ARE OMITTED.
2300      01 DISP-REC.
2400          COPY DDS-ALL-FORMATS OF CUSMING.
2500      FD CUST-MASTER
2600          LABEL RECORDS ARE STANDARD.
2700      01 CUST-REC.
2800          COPY DDS-CUSMST OF CUSMSTP.
2900      WORKING-STORAGE SECTION.
3000      01 ONE
3100      01 CM-STATUS
3200      01 WS-CONTROL.
3300          02 WS-IND
3400          02 WS-FORMAT
                                     PIC 1 VALUE 8"1".
                                     PIC XX.
                                     PIC XX.
                                     PIC X(10).

```

The SEU listing of the Identification, Environment, and Data Division statements for this sample program is shown here. In particular, note the FILE-CONTROL and FD entries.

Figure 7-11 (Part 4 of 8). Example of a TRANSACTION Inquiry Program Using a Single Display Device

COBOL SOURCE LISTING

STMT SEQNBR -A 1 B. . . . 2 3 4 5 6 7 .IDENTFCN S COPYNAME CHG/DATE

```

1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID. INQUIRY.
3 000300 ENVIRONMENT DIVISION.
4 000400 CONFIGURATION SECTION.
5 000500 SOURCE-COMPUTER. IBM-S38.
6 000600 OBJECT-COMPUTER. IBM-S38.
7 000700 INPUT-OUTPUT SECTION.
8 000800 FILE-CONTROL.
9 000900     SELECT CUST-DISPLAY
10 001000     ASSIGN TO WORKSTATION-CUSMINQ.
11 001100     ORGANIZATION IS TRANSACTION.
12 001200     CONTROL-AREA IS WS-CONTROL.
13 001300     SELECT CUST-MASTER
14 001400     ASSIGN TO DATABASE-CUSMSTP.
15 001500     ORGANIZATION IS INDEXED.
16 001600     ACCESS IS RANDOM
17 001700     RECORD KEY IS CUST OF CUSMST.
18 001800     FILE STATUS IS CM-STATUS.
19 001900 DATA DIVISION.
20 002000 FILE SECTION.
21 002100 FD CUST-DISPLAY
22 002200 LABEL RECORDS ARE OMITTED.
23 002300 01 DISP-REC.
24 002400 COPY DDS-ALL-FORMATS OF CUSMINQ.
26 +000001 05 CUSMINQ-RECORD PIC X(80).
27 +000002
+000003* INPUT FORMAT:CUSPMT FROM FILE CUSMINQ OF LIBRARY PUBS
+000004* CUSTOMER PROMPT
28 +000005 05 CUSPMT-I REDEFINES CUSMINQ-RECORD.
29 +000006 06 CUSPMT-I-INDIC.
30 +000007 07 IN15 PIC 1 INDIC 15.
+000008* END OF PROGRAM
31 +000009 07 IN99 PIC 1 INDIC 99.
+000010* CUSTOMER NUMBER NOT FOUND PRESS RESET. TH
32 +000011 06 CUST PIC X(15).
+000012* CUSTOMER NUMBER
33 +000013
+000014* OUTPUT FORMAT:CUSPMT FROM FILE CUSMINQ OF LIBRARY PUBS
+000015* CUSTOMER PROMPT
34 +000016 05 CUSPMT-O REDEFINES CUSMINQ-RECORD.
35 +000017 06 CUSPMT-O-INDIC.
36 +000018 07 IN99 PIC 1 INDIC 99.
+000019* CUSTOMER NUMBER NOT FOUND PRESS RESET. TH
37 +000020
+000021*
+000022* END OF: CUSPMT ->
+000023*
38 +000024
+000025* INPUT FORMAT:CUSFLDS FROM FILE CUSMINQ OF LIBRARY PUBS
+000026* CUSTOMER DISPLAY
39 +000027 05 CUSFLDS-I REDEFINES CUSMINQ-RECORD.
40 +000028 06 CUSFLDS-I-INDIC.
41 +000029 07 IN15 PIC 1 INDIC 15.
+000030* END OF PROGRAM
42 +000031
+000032* OUTPUT FORMAT:CUSFLDS FROM FILE CUSMINQ OF LIBRARY PUBS
+000033* CUSTOMER DISPLAY
43 +000034 05 CUSFLDS-O REDEFINES CUSMINQ-RECORD.
44 +000035 06 NAME PIC X(25).
+000036* CUSTOMER NAME
45 +000037 06 ADDR PIC X(20).
+000038* CUSTOMER ADDRESS
46 +000039 06 CITY PIC X(20).
+000040* CUSTOMER CITY
47 +000041 06 STATE PIC X(2).
+000042* STATE
48 +000043 06 ZIP PIC S9(5).
+000044* ZIP CODE
49 +000045 06 ARBAL PIC S9(6)V9(2).
+000046* ACCTS REC BALANCE
50 +000047
+000048*
+000049* END OF: CUSFLDS ->
+000050*

```

The Data Division for this sample program is shown after compilation to illustrate the data structures generated by the COPY statements, DDS formats.

Figure 7-11 (Part 5 of 8). Example of a TRANSACTION Inquiry Program Using a Single Display Device


```

51 002500 FD CUST-MASTER
52 002600 LABEL RECORDS ARE STANDARD.
53 002700 01 CUST-REC.
54 002800 COPY DDS-CUSMST OF CUSMSTP.
56 +000001
+000002* I-O FORMAT:CUSMST FROM FILE CUSMSTP OF LIBRARY PUBS CUSMST
+000003* CUSTOMER MASTER RECORD CUSMST
57 +000004 05 CUSMST. CUSMST
58 +000005 06 CUST PIC X(5). CUSMST
+000006* CUSTOMER NUMBER CUSMST
59 +000007 06 NAME PIC X(25). CUSMST
+000008* CUSTOMER NAME CUSMST
.60 +000009 06 ADDR PIC X(20). CUSMST
+000010* CUSTOMER ADDRESS CUSMST
61 +000011 06 CITY PIC X(20). CUSMST
+000012* CUSTOMER CITY CUSMST
62 +000013 06 STATE PIC X(2). CUSMST
+000014* STATE CUSMST
63 +000015 06 ZIP PIC S9(5) COMP-3. CUSMST
+000016* ZIP CODE CUSMST
64 +000017 06 SRHCOB PIC X(6). CUSMST
+000018* CUSTOMER NUMBER SEARCH CODE CUSMST
65 +000019 06 CUSTYP PIC S9(1) COMP-3. CUSMST
+000020* CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT CUSMST
66 +000021 06 ARBAL PIC S9(6)V9(2) COMP-3. CUSMST
+000022* ACCTS REC BALANCE CUSMST
67 +000023 06 ORDBAL PIC S9(6)V9(2) COMP-3. CUSMST
+000024* A/R AMT IN ORDER FILE CUSMST
68 +000025 06 LSTAMT PIC S9(6)V9(2) COMP-3. CUSMST
+000026* LAST AMOUNT PAID IN A/R CUSMST
69 +000027 06 LSTDAT PIC S9(6) COMP-3. CUSMST
+000028* LAST DATE PAID IN A/R CUSMST
70 +000029 06 CRDLMT PIC S9(6)V9(2) COMP-3. CUSMST
+000030* CUSTOMER CREDIT LIMIT CUSMST
71 +000031 06 SLSYR PIC S9(8)V9(2) COMP-3. CUSMST
+000032* CUSTOMER SALES THIS YEAR CUSMST
72 +000033 06 SLSLYR PIC S9(8)V9(2) COMP-3. CUSMST
+000034* CUSTOMER SALES LAST YEAR CUSMST
73 002900 WORKING-STORAGE SECT IDN.
74 003000 01 ONE PIC 1 VALUE B*1*.
75 003100 01 CM-STATUS PIC XX.
76 003200 01 WS-CONTROL.
77 003300 02 WS-IND PIC XX.
78 003400 02 WS-FORMAT PIC X(10).

```

The Data Division for this sample program is shown after compilation to illustrate the data structures generated by the COPY statements, DDS formats.

Figure 7-11 (Part 6 of 8). Example of a TRANSACTION Inquiry Program Using a Single Display Device

```

                                COBOL SOURCE LISTING                INQUIRY
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S  COPYNAME  CHG/DATE

79 003500 PROCEDURE DIVISION.
003600 BEGIN.
80 003700 OPEN I-O CUST-DISPLAY. INPUT CUST-MASTER.
81 003800 MOVE ZERO TO IN99 OF CUSPMT-O.
003900 LOOP.
82 004000 WRITE DISP-REC FORMAT IS "CUSPMT".
83 004100 READ CUST-DISPLAY RECORD.
84 004200 IF IN15 OF CUSPMT-I
004300 IS EQUAL TO ONE
85 004400 THEN GO TO FINIS.
86 004500 MOVE CUST OF CUSPMT-I TO CUST OF CUSMST.
87 004600 READ CUST-MASTER RECORD.
88 004700 IF CM-STATUS IS NOT EQUAL "00" THEN
89 004800 MOVE ONE TO IN99 OF CUSPMT-O. GO TO LOOP.
91 004900 MOVE CORRESPONDING CUSMST TO CUSFLDS-O.
92 005000 WRITE DISP-REC FORMAT IS "CUSFLDS".
93 005100 READ CUST-DISPLAY RECORD.
94 005200 IF IN15 OF CUSFLDS-I
005300 IS EQUAL TO ONE
95 005400 THEN GO TO FINIS.
96 005500 MOVE ZERO TO IN99 OF CUSPMT-O.
97 005600 GO TO LOOP.
005700 FINIS.
98 005800 CLOSE CUST-DISPLAY. CUST-MASTER.
005900 RETURN-TO-CALLER.
99 005000 EXIT PROGRAM.

```

The WRITE operation in statement 82 writes the CUSPMT record to the display. This record prompts the user to enter a customer number. If the user enters a customer number and presses the Enter key, the next READ operation then reads the record back into the program.

The READ operation in statement 87 uses the customer number (CUST) field to retrieve the corresponding CUSMST record from the CUSMSTP file. If no record is found in the CUSMSTP file, indicator 99 is set on. The GO TO operation in statement 89, which is executed when indicator 99 is set on, causes the program to branch back to the beginning. The message 'Customer number not found' is displayed because it is conditioned by indicator 99 in the DDS for the file, and the keyboard is locked. The user must press the Reset key in response to this message to unlock the keyboard. The user can then enter another customer number.

If the READ operation retrieves a record from the CUSMSTP file, the WRITE operation writes the CUSFLDS record to the display work station. This record contains the customer's name, address, and accounts receivable balance.

The user then presses the Enter key, and the program branches back to the beginning of the calculations. The user can enter another customer number or end the program. To end the program, the user presses the command attention key 01, which sets on indicator 15 in the program.

When indicator 15 is on, the program closes all files and executes the EXIT PROGRAM statement, which causes the program to return control to whoever called the COBOL program.

Figure 7-11 (Part 7 of 8). Example of a TRANSACTION Inquiry Program Using a Single Display Device

This is the initial display written by the WRITE operation in statement 82:

```
Customer Master Inquiry
Customer Number _____
Use CF1 to end program, use enter key to return to prompt screen
```

This display appears if a record is found in the CUSMSTP file for the customer number entered in response to the first display:

```
Customer Master Inquiry
Customer Number 10000
Use CF1 to end program, use enter key to return to prompt screen

Name    EXAMPLE WHOLESALERS
Address 3561 60TH STREET
City    MOLINE
State   IL           Zipcode 61265
A/R balance 137.02
```

This display appears if the CUSMSTP file does not contain a record for the customer number entered in response to the first display:

```
Customer Master Inquiry
Customer Number 10001
Use CF1 to end program, use enter key to return to prompt screen

Customer number not found press reset, then enter valid number
```

Figure 7-11 (Part 8 of 8). Example of a TRANSACTION Inquiry Program Using a Single Display Device

Figure 7-12 shows a sample order inquiry program, ORD220, that uses subfiles. The associated DDS is also shown, except for the DDS for the customer master file, CUSMSTP. Refer to Figure 7-11 for the DDS for CUSMSTP.

ORD220 displays all the detail order records for the requested order number. The program prompts the user to enter the order number that is to be reviewed. The order number is checked against the order header file, ORDHDRP. If the order number exists, the customer number accessed from the order header file is checked against the customer master file, CUSMSTP. All order detail records in ORDDTLP for the requested order are read and written to the subfile. A write for the subfile control record format is executed, and the detail order records in the subfile are displayed on the screen for the user to review. The program is ended by pressing command key 12.

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-1 UM/050*
Printed in U.S.A.

File	Keying Instruction	Graphic								Description	Page	of
Programmer	Date	Key										

Sequence Number	Form Type	And/Or Comment (A/O/*)	Conditioning				Name	Length	References (R)	Data Type (M/AP/S/B/A/S/X/Y/N/I/M)	Position	Usage (M/O//B/H/M)	Location		Functions
			Indicator	Not (N)	Indicator	Not (N)							Line	Pos	
1	A	K	K	K	PHYSICAL	ORDHDRP	ORDER					HEADER	FILE		
	A					ORDHDR								TEXT('Order Header Record')	
	A					CUST	5							TEXT('Customer number')	
	A					ORDER	5		00					TEXT('Order number')	
	A					ORDDAT	6		00					TEXT('Date order was entered')	
	A					CUSORD	15							TEXT('Customer purchase order + number')	
	A					SHPVIA	15							TEXT('Shipping instructions')	
	A					ORDSTS	1		00					TEXT('Order status 1=PCS 2=CNT 3=CHK + 4=RDY 5=SPRT 6=PCK')	
	A					OPRNAM	10							TEXT('Operator name who entered + the order')	
	A					ORDAMT	8		02					TEXT('Total dollar amount of + the order')	
	A					CUSTYP	1		00					TEXT('Customer type 1=GOV 2=SCH + 3=BUS 4=PVT 5=OT')	
	A					INVNUM	5		00					TEXT('Invoice number')	
	A					PRTDAT	6		00					TEXT('Date order was printed')	
	A					OPNSTS	1		00					TEXT('Order open status 1=OPEN + 2=CLOSE 3=CANCEL')	
	A					TOTLIN	3		00					TEXT('Total line items in order')	
	A					ACTMTH	2		00					TEXT('Accounting month of sale')	
	A					ACTYR	2		00					TEXT('Accounting year of sale')	
	A					STATE	2							TEXT('State')	
	A					AMPAID	8		02					TEXT('Total dollar amount paid')	
	A				K	ORDER									

*Number of sheets per pad may vary slightly.

Figure 7-12 (Part 1 of 14). Sample Order Inquiry Program

File	Keying Instruction	Graphic					
Programmer	Date	Key					

Description	Page	of
-------------	------	----

Sequence Number	Form Type And/Or Comment (A/O/*)	Conditioning				Name	Length	Reference (R)	Data Type (B A/P/S/B A/S/X/Y/N/I/M)	Decimal Positions	Usage (B/O/B/H/M)	Location		Functions
		Indicator Not (N)	Indicator Not (N)	Indicator Not (N)	Indicator Not (N)							Line	Pos	
1	A*					PHYSICAL ORDDTL								ORDER DETAIL FILE
2	A*													TEXT('Warehouse location')
3	A*					R ORDDTL								TEXT('Order detail record')
4	A*					CUST			5					CHECK(MF) COLHDG('Customer' 'Number')
5	A*					ORDER			5					COLHDG('Order' 'Number')
6	A*					LINNUM			3					COLHDG('Line' 'No.')
7	A*													TEXT('Line number of line in order' +)
8	A*					ITEM			5					CHECK(MIØ) COLHDG('Item' 'Number')
9	A*					QTYORD			3					COLHDG('Quantity' 'Ordered')
10	A*													TEXT('Quantity ordered')
11	A*					DESCRP			3Ø					COLHDG('Item description')
12	A*					PRICE			6	2				CMP(GT Ø) COLHDG('Price')
13	A*													TEXT('selling price')
14	A*					EXTENS			8	2				EDTCDE(J) COLHDG('Extension')
15	A*													TEXT('Extension amount of qtyord x + price')
16	A*					WHSLOC			3					CHECK(MF) COLHDG('Bin' 'No.')
17	A*					ORDDAT			6					TEXT('Date order was + entered')
18	A*					CUSTYP			1					RANGE(1 5) COLHDG('Cust' 'Type')
19	A*													TEXT('Customer type 1=GOV 2=SCH + 3=BUS 4=PVT 5=OT:')
20	A*					STATE			2					CHECK(MF) COLHDG('State')
21	A*					ACTMTH			2					COLHDG('Acct' 'Mth')
22	A*													TEXT('Accounting month of sale')
23	A*					ACTYR			2					COLHDG('Acct' 'Year')
24	A*													TEXT('Accounting year of sale')
25	A*					K ORDER								
26	A*					K LINNUM								

*Number of sheets per pad may vary slightly.

Figure 7-12 (Part 2 of 14). Sample Order Inquiry Program

File	Keying Instruction	Graphic	Description	Page	of
Programmer	Date	Key			

Sequence Number	Form Type	Form Code	Conditioning				Name	Length	Reference (R)	Data Type (B, A, P, S, B, A, S, A, T, I, N, U, M)	Location		Functions
			Condition Name	Indicator	Not (N)	Not (N)					Line	Pos	
A	H	ORD22										EXISTING ORDER REVIEW	
A	R											SFL	
A											10	TEXT('Item number')	
A											10	TEXT('Quantity ordered')	
A											10	TEXT('Item description')	
A											46	TEXT('Selling price')	
A											56	EDTCDE(J)	
A												TEXT('Extension amount of + qtyord x price')	
A	R	SUBCTL1										SFLCTL(SUBL)	
A												SFLCLR	
A												SFLDSP	
A												SFLDSPCTL	
A												SFLSIZ(S7)	
A												SFLPAG(14)	
A												SFLEND	
A												OVERLAY	
A												LOCK	
A												ROLLUP(97 'Continue display')	
A												CA12(98 'End of program')	
A												SETOFF(S7 'Display subfile')	
A												SETOFF(S8 'Off=Display subctl On=C1+ ear subfile')	
A												2 'Existing order inquiry'	
A												2 'Order'	
A												8 TEXT('Order number')	
A												ERRMSG('Order number not found' 61)	
A												ERRMSG('No lines for this order' 47)	
A												ERRMSG('No customer record found for r this order' 62)	
A												2 'Date'	
A												7 TEXT('Date order was entered')	
A												2 'Cust #'	
A												9 TEXT('Customer number')	
A												16 TEXT('Customer name')	
A												16 TEXT('Customer address')	
A												16 TEXT('Customer city')	
A												16 TEXT('State')	
A												31 TEXT('Zip code')	
A												44 'Total'	
A												51 TEXT('Total dollar amount of the + order')	
A												44 'Status'	
A												51	
A												44 'Open'	
A												51	
A												44 'Customer order'	
A												59 TEXT('Customer purchase order + number')	
A												44 'Ship via'	
A												59 TEXT('Shipping instructions')	
A												44 'Printed date'	
A												57 TEXT('Date order was printed')	
A												29 'Invoice'	
A												38 TEXT('Invoice number')	
A												64 'MTH'	
A												68 TEXT('Accounting month of sale')	
A												72 'Year'	
A												77 TEXT('Accounting year of sale')	
A												2 'Item'	
A												8 'Qty'	
A												14 'Item description'	
A												46 'Price'	
A												55 'Extension'	

*Number of lines per card may vary slightly

Figure 7-12 (Part 3 of 14). Sample Order Inquiry Program

COBOL SOURCE LISTING

STMT SEQNBR -A 1 B.. ... 2 3 4 5 6 7 .IDENTFCN 5 COPYNAME CHG/DATF

```

1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID.
3 000300   ORD220.
4 000400 ENVIRONMENT DIVISION.
5 000500 CONFIGURATION SECTION.
6 000600 SOURCE-COMPUTER.
7 000700   IBM-S38.
8 000800 OBJECT-COMPUTER.
9 000900   IBM-S38.
10 001000 INPUT-OUTPUT SECTION.
11 001100 FILE-CONTROL.
12 001200   SELECT ORDER-HEADER-FILE
13 001300     ASSIGN TO DATABASE-ORDHDRP
14 001400     ORGANIZATION IS INDEXED
15 001500     ACCESS MODE IS RANDOM
16 001600     RECORD KEY IS ORDER OF ORDER-HEADER-RECORD.
17 001700   SELECT ORDER-DETAIL-FILE
18 001800     ASSIGN TO DATABASE-ORDDTLP
19 001900     ORGANIZATION IS INDEXED
20 002000     ACCESS MODE IS DYNAMIC
21 002100     RECORD KEY IS ORDER-DETAIL-RECORD-KEY.
22 002200   SELECT CUSTOMER-MASTER-FILE
23 002300     ASSIGN TO DATABASE-CUSMSTP
24 002400     ORGANIZATION IS INDEXED
25 002500     ACCESS MODE IS RANDOM
26 002600     RECORD KEY IS CUST OF CUSTOMER-MASTER-RECORD.
27 002700   SELECT EXISTING-ORDER-DISPLAY-FILE
28 002800     ASSIGN TO WORKSTATION-ORD220D
29 002900     ORGANIZATION IS TRANSACTION
30 003000     ACCESS MODE IS DYNAMIC
31 003100     RELATIVE KEY IS SUBFILE-RECORD-NUMBER
32 003200     FILE STATUS IS STATUS-CODE-ONE.
33 003300 DATA DIVISION.
34 003400 FILE SECTION.
35 003500 FD ORDER-HEADER-FILE
36 003600   LABEL RECORDS ARE STANDARD.
37 003700 01 ORDER-HEADER-RECORD.
38 003800   COPY DDS-ORDHDR OF ORDHDRP.
40 *000001                                ORDHDR
*000002* I-O FORMAT:ORDHDR      FROM FILE ORDHDRP      OF LIBRARY CBL2849      ORDHDR
*000003*                                ORDER HEADER RECORD      ORDHDR
*000004*THE KEY DEFINITIONS FOR RECORD FORMAT ORDHDR      ORDHDR
*000005* NUMBER          NAME          RETRIEVAL          TYPE          ALTSEQ      ORDHDR
*000006* 0001          ORDER          ASCENDING          N          NO          ORDHDR
41 *000007          05 ORDHDR.          ORDHDR
42 *000008          06 CUST          PIC X(5).          ORDHDR
*000009*          CUSTOMER NUMBER          ORDHDR
43 *000010          06 ORDER          PIC S9(5)          COMP-3.          ORDHDR
*000011*          ORDER NUMBER          ORDHDR
44 *000012          06 ORDUAT          PIC S9(6)          COMP-3.          ORDHDR
*000013*          DATE ORDER WAS ENTERED          ORDHDR
45 *000014          06 CUSORD          PIC X(15).          ORDHDR
*000015*          CUSTOMER PURCHASE ORDER NUMBER          ORDHDR
46 *000016          06 SHPVIA          PIC X(15).          ORDHDR
*000017*          SHIPPING INSTRUCTIONS          ORDHDR

```

Figure 7-12 (Part 4 of 14). Sample Order Inquiry Program

```

COBOL SOURCE LISTING                                ORD220
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN 5 COPYNAME CHG/DATE

47 *000018          06 ORDSTS          PIC S9(1)          COMP-3.          ORDHDR
   *000019*          ORDER STATUS IPCS 2CNT 3CHK 4ROY 5PRT 6PCK          ORDHDR
48 *000020          06 OPRNAM          PIC X(10).        ORDHDR
   *000021*          OPERATOR NAME WHO ENTERED THE ORDER          ORDHDR
49 *000022          06 ORDAMT          PIC S9(6)V9(2)    COMP-3.          ORDHDR
   *000023*          TOTAL DOLLAR AMOUNT OF THE ORDER          ORDHDR
50 *000024          06 CUSTYP          PIC S9(1)          COMP-3.          ORDHDR
   *000025*          CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT          ORDHDR
51 *000026          06 INVNUM          PIC S9(5)          COMP-3.          ORDHDR
   *000027*          INVOICE NUMBER          ORDHDR
52 *000028          06 PRDAT          PIC S9(6)          COMP-3.          ORDHDR
   *000029*          DATE ORDER WAS PRINTED          ORDHDR
53 *000030          06 OPNSTS          PIC S9(1)          COMP-3.          ORDHDR
   *000031*          ORDER OPEN STATUS 1=OPEN 2=CLOSE 3=CANCEL          ORDHDR
54 *000032          06 TOTLIN          PIC S9(3)          COMP-3.          ORDHDR
   *000033*          TOTAL LINE ITEMS IN ORDER          ORDHDR
55 *000034          06 ACTMTH          PIC S9(2)          COMP-3.          ORDHDR
   *000035*          ACCOUNTING MONTH OF SALE          ORDHDR
56 *000036          06 ACTYR          PIC S9(2)          COMP-3.          ORDHDR
   *000037*          ACCOUNTING YEAR OF SALE          ORDHDR
57 *000038          06 STATE          PIC X(2).         ORDHDR
   *000039*          STATE          ORDHDR
58 *000040          06 AMPAID          PIC S9(6)V9(2)    COMP-3.          ORDHDR
   *000041*          TOTAL DOLLAR AMOUNT PAID          ORDHDR
59 003900 FD ORDER-DETAIL-FILE
60 004000 LABEL RECORDS ARE STANDARD.
61 004100 01 ORDER-DETAIL-RECORD.
62 004200 COPY DDS-ORDDTL OF ORDDTLP.
64 *000001          ORDDTL          FROM FILE ORDDTLP OF LIBRARY CBL2849          ORDDTL
   *000002*          I-O FORMAT:ORDDTL          ORDER DETAIL RECORD          ORDDTL
   *000003*          THE KEY DEFINITIONS FOR RECORD FORMAT ORDDTL          ORDDTL
   *000004*          NUMBER          NAME          RETRIEVAL          TYPE          ALTSEQ          ORDDTL
   *000005*          0001          ORDER          ASCENDING          N          NO          ORDDTL
   *000006*          0002          LINNUM          ASCENDING          N          NO          ORDDTL
65 *000008          05 ORDDTL.          ORDDTL
66 *000009          06 CUST          PIC X(5).         ORDDTL
   *000010*          Customer number          ORDDTL
67 *000011          06 ORDER          PIC S9(5)          COMP-3.          ORDDTL
   *000012*          Order number          ORDDTL
68 *000013          06 LINNUM          PIC S9(3)          COMP-3.          ORDDTL
   *000014*          LINE NUMBER OF LINE IN ORDER          ORDDTL
69 *000015          06 ITEM          PIC S9(5)          COMP-3.          ORDDTL
   *000016*          Item number          ORDDTL
70 *000017          06 QTYORD          PIC S9(3)          COMP-3.          ORDDTL
   *000018*          QUANTITY ORDERED          ORDDTL
71 *000019          06 DESCRP          PIC X(30).        ORDDTL
   *000020*          Item description          ORDDTL
72 *000021          06 PRICE          PIC S9(4)V9(2)    COMP-3.          ORDDTL
   *000022*          SELLING PRICE          ORDDTL
73 *000023          06 EXTENS          PIC S9(6)V9(2)    COMP-3.          ORDDTL
   *000024*          EXTENSION AMOUNT OF QTYORD X PRICE          ORDDTL
74 *000025          06 WHSLDC          PIC X(3).         ORDDTL
   *000026*          Bin no.          ORDDTL
75 *000027          06 ORDDAT          PIC S9(6)          COMP-3.          ORDDTL

```

Figure 7-12 (Part 5 of 14). Sample Order Inquiry Program


```

COBOL SOURCE LISTING                                ORD220
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN 5 COPYNAME CHG/DATE

+000028*      DATE ORDER WAS ENTERED                                DRDDTL
76 +000029*      06 CUSTYP      PIC S9(1)      COMP-3.              DRDDTL
+000030*      CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT          DRDDTL
77 +000031*      06 STATE      PIC X(2).              DRDDTL
+000032*      State                                                DRDDTL
78 +000033*      06 ACTMTH     PIC S9(2)      COMP-3.              DRDDTL
+000034*      ACCOUNTING MONTH OF SALE                            DRDDTL
79 +000035*      06 ACTYR     PIC S9(2)      COMP-3.              DRDDTL
+000036*      ACCOUNTING YEAR OF SALE                             DRDDTL
80 004300 66 ORDER-DETAIL-RECORD-KEY RENAMES ORDER THRU LINNUM.
81 004400 FD CUSTOMER-MASTER-FILE
82 004500 LABEL RECORDS ARE STANDARD.
83 004600 01 CUSTOMER-MASTER-RECORD.
84 004700 COPY DDS-CUSMST OF CUSMSTP.
86 +000001*      CUSMST
+000002*      I-D FORMAT:CUSMST FROM FILE CUSMSTP OF LIBRARY CBL2849 CUSMST
+000003*      CUSTOMER MASTER RECORD CUSMST
+000004* THE KEY DEFINITIONS FOR RECORD FORMAT CUSMST CUSMST
+000005* NUMBER NAME RETRIEVAL TYPE ALTSEQ CUSMST
+000006* 0001 CUST ASCENDING AN NO CUSMST
87 +000007* 05 CUSMST. CUSMST
88 +000008* 06 CUST PIC X(5). CUSMST
+000009* CUSTOMER NUMBER CUSMST
89 +000010* 06 NAME PIC X(25). CUSMST
+000011* CUSTOMER NAME CUSMST
90 +000012* 06 ADDR PIC X(20). CUSMST
+000013* CUSTOMER ADDRESS CUSMST
91 +000014* 06 CITY PIC X(20). CUSMST
+000015* CUSTOMER CITY CUSMST
92 +000016* 06 STATE PIC X(2). CUSMST
+000017* STATE CUSMST
93 +000018* 06 ZIP PIC S9(5) COMP-3. CUSMST
+000019* ZIP CODE CUSMST
94 +000020* 06 SRHCOD PIC X(6). CUSMST
+000021* CUSTOMER NUMBER SEARCH CODE CUSMST
95 +000022* 06 CUSTYP PIC S9(1) COMP-3. CUSMST
+000023* CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT CUSMST
96 +000024* 06 ARBAL PIC S9(6)V9(2) COMP-3. CUSMST
+000025* ACCTS REC BALANCE CUSMST
97 +000026* 06 ORDBAL PIC S9(6)V9(2) COMP-3. CUSMST
+000027* A/R AMT IN ORDER FILE CUSMST
98 +000028* 06 LSTAMT PIC S9(6)V9(2) COMP-3. CUSMST
+000029* LAST AMOUNT PAID IN A/R CUSMST
99 +000030* 06 LSTDAT PIC S9(6) COMP-3. CUSMST
+000031* LAST DATE PAID IN A/R CUSMST
100 +000032* 06 CRDLMT PIC S9(6)V9(2) COMP-3. CUSMST
+000033* CUSTOMER CREDIT LIMIT CUSMST
101 +000034* 06 SLSYR PIC S9(8)V9(2) COMP-3. CUSMST
+000035* CUSTOMER SALES THIS YEAR CUSMST
102 +000036* 06 SLSLYR PIC S9(8)V9(2) COMP-3. CUSMST
+000037* CUSTOMER SALES LAST YEAR CUSMST
103 004800 FD EXISTING-ORDER-DISPLAY-FILE
104 004900 LABEL RECORDS ARE OMITTED.
105 005000 01 EXISTING-ORDER-DISPLAY-RECORD.
106 005100 COPY DDS-ALL-FORMATS OF ORD220D.

```

Figure 7-12 (Part 6 of 14). Sample Order Inquiry Program

```

                                COBOL SOURCE LISTING                                ORD220
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN 5 COPYNAME CHG/DATE

108 +000001      05  ORD220D-RECORD PIC X(171).                                <-ALL-FMTS
109 +000002      <-ALL-FMTS
   +000003#      I-O FORMAT:SUB1      FROM FILE ORD220D      OF LIBRARY CBL2849  <-ALL-FMTS
   +000004#      <-ALL-FMTS
110 +000005      05  SUB1      REDEFINES ORD220D-RECORD.                    <-ALL-FMTS
111 +000006      06  ITEM      PIC S9(5).                                <-ALL-FMTS
   +000007#      ITEM NUMBER      <-ALL-FMTS
112 +000008      06  QTYORD      PIC S9(3).                                <-ALL-FMTS
   +000009#      QUANTITY ORDERED      <-ALL-FMTS
113 +000010      06  DESCRP      PIC X(30).                                <-ALL-FMTS
   +000011#      ITEM DESCRIPTION      <-ALL-FMTS
114 +000012      06  PRICE      PIC S9(4)V9(2).                          <-ALL-FMTS
   +000013#      SELLING PRICE      <-ALL-FMTS
115 +000014      06  EXTENS      PIC S9(6)V9(2).                          <-ALL-FMTS
   +000015#      EXTENSION AMOUNT OF QTYORD X PRICE      <-ALL-FMTS
116 +000016      INPUT FORMAT:SUBCTL1      FROM FILE ORD220D      OF LIBRARY CBL2849 <-ALL-FMTS
   +000017#      <-ALL-FMTS
   +000018#      <-ALL-FMTS
117 +000019      05  SUBCTL1-I      REDEFINES ORD220D-RECORD.                <-ALL-FMTS
118 +000020      06  SUBCTL1-I-INDIC.      <-ALL-FMTS
119 +000021      07  IN97      PIC 1 INDIC 97.                            <-ALL-FMTS
   +000022#      CONTINUE DISPLAY      <-ALL-FMTS
120 +000023      07  IN98      PIC 1 INDIC 98.                            <-ALL-FMTS
   +000024#      END OF PROGRAM      <-ALL-FMTS
121 +000025      07  IN57      PIC 1 INDIC 57.                            <-ALL-FMTS
   +000026#      DISPLAY SUBFILE      <-ALL-FMTS
122 +000027      07  IN58      PIC 1 INDIC 58.                            <-ALL-FMTS
   +000028#      OFF=DISPLAY SUBCTL ON=CLEAR SUBFILE      <-ALL-FMTS
123 +000029      07  IN61      PIC 1 INDIC 61.                            <-ALL-FMTS
   +000030#      Order number not found      <-ALL-FMTS
124 +000031      07  IN47      PIC 1 INDIC 47.                            <-ALL-FMTS
   +000032#      No lines for this order      <-ALL-FMTS
125 +000033      07  IN62      PIC 1 INDIC 62.                            <-ALL-FMTS
   +000034#      No customer record found for this order      <-ALL-FMTS
126 +000035      06  ORDER      PIC S9(5).                                <-ALL-FMTS
   +000036#      ORDER NUMBER      <-ALL-FMTS
127 +000037      OUTPUT FORMAT:SUBCTL1      FROM FILE ORD220D      OF LIBRARY CBL2849 <-ALL-FMTS
   +000038#      <-ALL-FMTS
   +000039#      <-ALL-FMTS
128 +000040      05  SUBCTL1-O      REDEFINES ORD220D-RECORD.                <-ALL-FMTS
129 +000041      06  SUBCTL1-O-INDIC.      <-ALL-FMTS
130 +000042      07  IN58      PIC 1 INDIC 58.                            <-ALL-FMTS
   +000043#      OFF=DISPLAY SUBCTL ON=CLEAR SUBFILE      <-ALL-FMTS
131 +000044      07  IN57      PIC 1 INDIC 57.                            <-ALL-FMTS
   +000045#      DISPLAY SUBFILE      <-ALL-FMTS
132 +000046      07  IN45      PIC 1 INDIC 45.                            <-ALL-FMTS
133 +000047      07  IN47      PIC 1 INDIC 47.                            <-ALL-FMTS
   +000048#      No lines for this order      <-ALL-FMTS
134 +000049      07  IN61      PIC 1 INDIC 61.                            <-ALL-FMTS
   +000050#      Order number not found      <-ALL-FMTS
135 +000051      07  IN62      PIC 1 INDIC 62.                            <-ALL-FMTS
   +000052#      No customer record found for this order      <-ALL-FMTS
136 +000053      06  ORDER      PIC S9(5).                                <-ALL-FMTS
   +000054#      ORDER NUMBER      <-ALL-FMTS
137 +000055      06  ORDDAT      PIC S9(6).                                <-ALL-FMTS

```

Figure 7-12 (Part 7 of 14). Sample Order Inquiry Program

STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 IDENTFCN S COPYNAME CHG/DATE

```

+000056*      DATE ORDER WAS ENTERED          <-ALL-FMTS
138 +000057      06 CUST                        PIC X(5).          <-ALL-FMTS
+000058*      CUSTOMER NUMBER                  <-ALL-FMTS
139 +000059      06 NAME                        PIC X(25).         <-ALL-FMTS
+000060*      CUSTOMER NAME                    <-ALL-FMTS
140 +000061      06 ADDR                        PIC X(20).         <-ALL-FMTS
+000062*      CUSTOMER ADDRESS                 <-ALL-FMTS
141 +000063      06 CITY                        PIC X(20).         <-ALL-FMTS
+000064*      CUSTOMER CITY                    <-ALL-FMTS
142 +000065      06 STATE                       PIC X(2).          <-ALL-FMTS
+000066*      STATE                            <-ALL-FMTS
143 +000067      06 ZIP                         PIC S9(5).         <-ALL-FMTS
+000068*      ZIP CODE                          <-ALL-FMTS
144 +000069      06 ORDAMT                      PIC S9(6)V9(2).   <-ALL-FMTS
+000070*      TOTAL DOLLAR AMOUNT OF THE ORDER <-ALL-FMTS
145 +000071      06 STSORD                      PIC X(12).        <-ALL-FMTS
146 +000072      06 STSOPN                      PIC X(12).        <-ALL-FMTS
147 +000073      06 CUSORD                      PIC X(15).        <-ALL-FMTS
+000074*      CUSTOMER PURCHASE ORDER NUMBER <-ALL-FMTS
148 +000075      06 SHPVIA                      PIC X(15).        <-ALL-FMTS
+000076*      SHIPPING INSTRUCTIONS           <-ALL-FMTS
149 +000077      06 PRDAT                       PIC S9(6).        <-ALL-FMTS
+000078*      DATE ORDER WAS PRINTED         <-ALL-FMTS
150 +000079      06 INVNUM                      PIC S9(5).        <-ALL-FMTS
+000080*      INVOICE NUMBER                  <-ALL-FMTS
151 +000081      06 ACTMTH                      PIC S9(2).        <-ALL-FMTS
+000082*      ACCOUNTING MONTH OF SALE       <-ALL-FMTS
152 +000083      06 ACTYR                      PIC S9(2).        <-ALL-FMTS
+000084*      ACCOUNTIN YEAR OF SALE         <-ALL-FMTS
153 005200 WORKING-STORAGE SECTION.
154 005300 01 EXISTING-ORDER-DISPLAY-KEY.
155 005400 05 SUBFILE-RECORD-NUMBER          PIC 9(02)
156 005500                                VALUE ZERO.
157 005600 01 ORDER-STATUS-COMMENT-VALUES.
158 005700 05 FILLER                          PIC X(12)
159 005800     VALUE "1-IN PROCESS".
160 005900 05 FILLER                          PIC X(12)
161 006000     VALUE "2-CONTINUED ".
162 006100 05 FILLER                          PIC X(12)
163 006200     VALUE "3-CREDIT CHK".
164 006300 05 FILLER                          PIC X(12)
165 006400     VALUE "4-READY PRT ".
166 006500 05 FILLER                          PIC X(12)
167 006600     VALUE "5-PRINTED  ".
168 006700 05 FILLER                          PIC X(12)
169 006800     VALUE "6-PICKED  ".
170 006900 05 FILLER                          PIC X(12)
171 007000     VALUE "7-INVOCED  ".
172 007100 05 FILLER                          PIC X(12)
173 007200     VALUE "8-INVALID  ".
174 007300 05 FILLER                          PIC X(12)
175 007400     VALUE "9-CANCELED ".
176 007500 01 ORDER-STATUS-COMMENT-TABLE
177 007600     REDEFINES ORDER-STATUS-COMMENT-VALUES.
178 007700 05 ORDER-STATUS OCCURS 9 TIMES.
    
```

Figure 7-12 (Part 8 of 14). Sample Order Inquiry Program

		COBOL SOURCE LISTING					ORD220														
STMT	SEQNBR	-A	1	B..	...	2	...	3	...	4	...	5	...	6	...	7	.IDENTFCN	S	COPYNAME	CHG/DATE	
179	007800					10		ORDER-STATUS-COMMENT													
180	007900	01				OPEN-STATUS-COMMENT-VALUES.															
181	008000		05			FILLER															
182	008100					VALUE "1-OPEN	".														11/11/80
183	008200		05			FILLER															
184	008300					VALUE "2-CLOSED	".														11/11/80
185	008400		05			FILLER															
186	008500					VALUE "3-CANCELED	".														11/11/80
187	008600	01				OPEN-STATUS-COMMENT-TABLE															
188	008700					REDEFINES OPEN-STATUS-COMMENT-VALUES.															
189	008800		05			OPEN-STATUS OCCURS 3 TIMES.															
190	008900					10 OPEN-STATUS-COMMENT															
191	009000	01				ERRHDL-PARAMETERS.															
192	009100		05			STATUS-CODE-ONE															11/07/80
193	009200					88 SUBFILE-IS-FULL															11/11/80
194	009300	01				ERRPGM-PARAMETERS.															
195	009400		05			DISPLAY-PARAMETER															
196	009500																				
197	009600		05			DUMMY-ONE															
198	009700																				
199	009800		05			DUMMY-TWO															
200	009900																				
201	010000		05			STATUS-CODE-TWO.															
202	010100					10 PRIMARY															
203	010200					10 SECONDARY															
204	010300					10 FILLER															
205	010400																				
206	010500		05			DUMMY-THREE															
207	010600																				
208	010700	01				SWITCH-AREA.															
209	010800		05			SW01															
210	010900					88 NO-MORE-DETAIL-LINE-ITEMS															
211	011000					88 MORE-DETAIL-LINE-ITEMS-EXIST															
212	011100		05			SW02															
213	011200					88 WRITE-DISPLAY															
214	011300					88 READ-DISPLAY															
215	011400		05			SW03															
216	011500					88 SUBCTLI-FORMAT															
217	011600					88 NOT-SUBCTLI-FORMAT															
218	011700		05			SW04															
219	011800					88 SUB1-FORMAT															
220	011900					88 NOT-SUB1-FORMAT															
221	012000	01				INDICATOR-AREA.															
222	012100		05			IN98															
223	012200					88 END-OF-EXISTING-ORDER-INQUIRY															
224	012300		05			IN97															
225	012400					88 CONTINUE-DETAIL-LINES-DISPLAY															
226	012500		05			IN62															
227	012600					88 CUSTOMER-NOT-FOUND															
228	012700					88 CUSTOMER-EXIST															
229	012800		05			IN61															
230	012900					88 ORDER-NOT-FOUND															
231	013000					88 ORDER-EXIST															
232	013100		05			IN58															
233	013200					88 CLEAR-SUBFILE															

Figure 7-12 (Part 9 of 14). Sample Order Inquiry Program

COBOL SOURCE LISTING		ORD220	
STMT	SEQNBR	-A 1 B...	2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME CHG/DATE
234	013300	88 DISPLAY-SUBFILE-CONTROL	VALUE B'0'.
235	013400	05 IN57	PIC 1 INDIC 57.
236	013500	88 DISPLAY-SUBFILE	VALUE B'1'.
237	013600	05 IN47	PIC 1 INDIC 47.
238	013700	88 NO-DETAIL-LINES-FOR-ORDER	VALUE B'1'.
239	013800	88 DETAIL-LINES-FOR-ORDER-EXIST	VALUE B'0'.
240	013900	05 IN45	PIC 1 INDIC 47.
241	014000	88 END-OF-ORDER	VALUE B'1'.
242	014100	PROCEDURE DIVISION.	
	014200	DECLARATIVES.	
	014300	TRANSACTION-ERROR SECTION.	
	014400	USE AFTER STANDARD ERROR PROCEDURE	
	014500	EXISTING-ORDER-DISPLAY-FILE.	
	014600	WORK-STATION-ERROR-HANDLER.	
243	014700	IF SUBFILE-IS-FULL THEN NEXT SENTENCE	11/07/80
	014800	ELSE	11/07/80
244	014900	DISPLAY 'WORK-STATION ERROR' STATUS-CODE-ONE.	11/07/80
	015000	END DECLARATIVES.	
	015100	INQUIRY-INTO-EXISTING-ORDER SECTION.	
	015200	MAINLINE-ROUTINE.	
245	015300	PERFORM SET-UP-ROUTINE.	
246	015400	PERFORM EXISTING-ORDER-INQUIRY	
	015500	UNTIL END-OF-EXISTING-ORDER-INQUIRY.	
247	015600	PERFORM CLEAN-UP-ROUTINE.	
	015700	SET-UP-ROUTINE.	
248	015800	OPEN INPUT ORDER-HEADER-FILE	
	015900	ORDER-DETAIL-FILE	
	016000	CUSTOMER-MASTER-FILE	
	016100	I-O EXISTING-ORDER-DISPLAY-FILE.	
249	016200	MOVE SPACES TO CUST OF SUBCTLI-O	
	016300	NAME OF SUBCTLI-O	
	016400	ADDR OF SUBCTLI-O	
	016500	CITY OF SUBCTLI-O	
	016600	STATE OF SUBCTLI-O	
	016700	STSDRD OF SUBCTLI-O	
	016800	STSDPN OF SUBCTLI-O	
	016900	CUSORD OF SUBCTLI-O.	
250	017000	MOVE ZEROS TO ORDER OF SUBCTLI-O	11/07/80
	017100	ORDDAT OF SUBCTLI-O	
	017200	ZIP OF SUBCTLI-O	
	017300	URDAMT OF SUBCTLI-O	
	017400	PRTDAT OF SUBCTLI-O	
	017500	INVNUM OF SUBCTLI-O	
	017600	ACTMTH OF SUBCTLI-O	
	017700	ACTYR OF SUBCTLI-O.	
251	017800	MOVE B'0' TO INDICATOR-AREA.	
252	017900	SET READ-DISPLAY	
	018000	NOT-SUBCTLI-FORMAT	
	018100	NOT-SUB1-FORMAT TO TRUE.	
253	018200	MOVE CORR INDICATOR-AREA TO SUBCTLI-O-INDIC.	
254	018300	SET WRITE-DISPLAY, SUBCTLI-FORMAT TO TRUE.	
255	018400	WRITE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS 'SUBCTLI'.	
256	018500	READ EXISTING-ORDER-DISPLAY-FILE RECORD.	
257	018600	MOVE CORR SUBCTLI-I-INDIC TO INDICATOR-AREA.	
	018700	EXISTING-ORDER-INQUIRY.	

Figure 7-12 (Part 10 of 14). Sample Order Inquiry Program

STMT	SEQNBR	-A	1	B..	...	2	...	3	...	4	...	5	...	6	...	7	.IDENTFCN	5	COPYNAME	CHG/DATE
258	018800																			
259	018900																			
260	019000																			
261	019100																			
262	019200																			
262	019300																			
	019400																			
	019500																			
263	019600																			
	019700																			
264	019800																			
	019900																			
	020000																			
265	020100																			
266	020200																			
267	020300																			
268	020400																			
269	020500																			
270	020600																			
	020700																			
271	020800																			
272	020900																			
273	021000																			
274	021100																			
275	021200																			
276	021300																			
277	021400																			
	021500																			
	021600																			
	021700																			
	021800																			
	021900																			
	022000																			
	022100																			
278	022200																			
	022300																			
279	022400																			
	022500																			
280	022600																			
	022700																			
281	022800																			
	022900																			
282	023000																			
	023100																			
283	023200																			
	023300																			
284	023400																			
	023500																			
285	023600																			
286	023700																			
287	023800																			
	023900																			
288	024000																			
289	024100																			
290	024200																			

11/07/80
11/07/80

Figure 7-12 (Part 11 of 14). Sample Order Inquiry Program

```

                                COBOL SOURCE LISTING                                ORG220
STMT SEQNBR -A 1 B... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 .IDENTFCN S  COPYNAME  CHG/DATE

291 024300    WRITE EXISTING-ORDER-DISPLAY-RECORD
    024400        FORMAT IS 'SUBCTL1'.
292 024500    SET DISPLAY-SUBFILE-CONTROL TO TRUE.
293 024600    PERFORM BUILD-DISPLAY-SUBFILE
    024700        UNTIL NO-MORE-DETAIL-LINE-ITEMS
    024800        OR SUBFILE-IS-FULL.                                     11/07/80
294 024900    MOVE CORR ORDHDR OF ORDER-HEADER-RECORD
    025000        TO SUBCTL1-0 OF EXISTING-ORDER-DISPLAY-RECORD.
295 025100    MOVE CORR CUSMST OF CUSTOMER-MASTER-RECORD
    025200        TO SUBCTL1-0 OF EXISTING-ORDER-DISPLAY-RECORD .      11/06/80
296 025300    MOVE ORDER-STATUS(ORDSTS) TO STSGRD.
297 025400    MOVE OPEN-STATUS(OPNSTS) TO STSOPN.
298 025500    SET MORE-DETAIL-LINE-ITEMS-EXIST TO TRUE.
299 025600    MOVE ZEROS TO SUBFILE-RECORD-NUMBER.
    025700 BUILD-DISPLAY-SUBFILE.                                     11/07/80
300 025800    MOVE CORR ORDRTL OF ORDER-DETAIL-RECORD
    025900        TO SUB1 OF EXISTING-ORDER-DISPLAY-RECORD.
301 026000    SET WRITE-DISPLAY, SUB1-FORMAT TO TRUE.
302 026100    ADD 1 TO SUBFILE-RECORD-NUMBER.
303 026200    WRITE SUBFILE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS 'SUB1'.
304 026300    IF SUBFILE-IS-FULL THEN
305 026400        SET DISPLAY-SUBFILE TO TRUE
    026500    ELSE
    026600        PERFORM READ-NEXT-ORDER-DETAIL-RECORD
    026700        IF NO-MORE-DETAIL-LINE-ITEMS THEN
    026800            NEXT SENTENCE
    026900        ELSE
308 027000            IF ORDER OF ORDER-DETAIL-RECORD IS NOT EQUAL TO
    027100                ORDER OF ORDER-HEADER-RECORD THEN
309 027200                SET DISPLAY-SUBFILE,
    027300                    NO-MORE-DETAIL-LINE-ITEMS TO TRUE
    027400            ELSE
    027500                NEXT SENTENCE.
    027600 READ-NEXT-ORDER-DETAIL-RECORD.
310 027700    READ ORDER-DETAIL-FILE NEXT RECORD
311 027800        AT END SET DISPLAY-SUBFILE,
    027900            NO-MORE-DETAIL-LINE-ITEMS TO TRUE.
    028000 CLEAN-UP-ROUTINE.
312 028100    CLOSE    ORDER-HEADER-FILE
    028200            ORDER-DETAIL-FILE
    028300            CUSTOMER-MASTER-FILE
    028400            EXISTING-ORDER-DISPLAY-FILE.
313 028500    STOP RUN.

                                * * * * *  E N D  O F  S O U R C E  * * * * *

```

Figure 7-12 (Part 12 of 14). Sample Order Inquiry Program

This is the initial order entry prompt display written to the workstation:

```

Existing Order Inquiry                Total 00000000
Order 00000                          Status
Date 000000                          Open
Cust #                               Customer order
                                Ship via
                                Printed date 000000
                                Mth 00 Year 00
Invoice 00000
Item Qty  Item description           Price  Extension
  
```

This display appears if there are detail order records for the customer whose order number was entered in the first display:

```

Existing Order Inquiry                Total 00742656
Order 17924  TESTCASE HARDWARE CO    Status 7-INVOICED
Date 110580  1204 BURNSIDE DR        Open 2-CLOSED
Cust # 11200  KANKAKEE                Customer order TESTCS17933001I
                                Ship via TRUCKCO
                                Printed date 042578
                                Mth 12 Year 78
Invoice 60901 17924
Item Qty  Item description           Price  Extension
33001 003 TORQUE WRENCH 75LB 14 INCH 009115 273.45
33100 001 TORQUE WRENCH W/GUAGE 200 LB 015777 157.77
44529 004 WOOD CHISEL - 3 1/4        006840 273.60
44958 002 POWER DRILL - 3/8 REV     008200 164.00
46102 003 WROUGHT IRON RAILING 4FTX6FT 007930 237.90
46201 001 WROUGHT IRON HAND RAIL 4X4FT 007178 71.78
47902 005 ESCUTCHEON BRASS 15X4INCHES 044488 2,224.40
48108 002 DOOR CHIME ELECTRIC 6 NOTE 104202 2,084.04
48801 004 AWNING ALUMINUM 4FT STRIPED 043002 1,720.08
48900 001 AWNING FIBERGLASS STRIPED 6FT 021954 219.54
  
```

Figure 7-12 (Part 13 of 14). Sample Order Inquiry Program

This display appears if the ORDHDRP file does not contain a record for the order number entered on the first display:

Existing Order Inquiry		Total	00000000	
Order	12400	Status		
Date	000000	Open		
Cust #		Customer order		
		Ship via		
	00000	Printed date	000000	
Invoice	00000	Mth	00	
		Year	00	
Item	Qty	Item description	Price	Extension

Order number not found

Figure 7-12 (Part 14 of 14). Sample Order Inquiry Program

Figure 7-13 shows a sample payment update program, ARC010, with the related DDS and sample display screens. For the DDS for the customer master file, CUSMSTP, refer to Figure 7-11.

In this example, payments from customers are registered. The clerk is prompted to enter one or more customer numbers and the amount of money to be credited to each customer's account. The program checks the customer number and unconditionally accepts any payment for an existing customer who has invoices outstanding. If an overpayment will result from the amount of the payment from a customer, the clerk is given the option to accept or reject the payment. If no customer record exists for a customer number, an error message is issued. Payments can be entered until the clerk ends the program by pressing command key 12.

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-1 UM/050*
Printed in U.S.A.

File		Keying Instruction	Graphic					Description	Page	of
Programmer	Date		Key							
A										
Sequence Number	Form Type 7 A=Add/Comment (A/O/)* 8 Not (N)	Conditioning Condition Name 9 Indicator 10 Not (N) 11 Indicator 12 Not (N) 13 Indicator 14 Not (N) 15 Indicator 16 Name Type (M/R/K/S/OI) 17 Reserved	Name	Length	Reference (R)	Data Type (A/P/S/B/A/S/X/Y/N/I/W) 35 Decimal 36 Positions 37 Usage (M/O/I/B/H/W)	Location Line 38-39 40-41 42-43 44	Pos	Functions	
1	A*	LOGICAL	ORDHDR			ORDER	FILE OF	ORDHDR		
2	A		R	ORDHDR				PFILE(ORDHDRP)		
3	A*									
4	A		CUST							
5	A		INVNUM							
6	A		ORDER							
7	A		ORDDAT							
8	A		CUSORD							
9	A		SHPVIA							
10	A		ORDSTS							
11	A		OPRNAM							
12	A		ORDAMT							
13	A		CUSTYP							
14	A		PRTDAT							
15	A		OPNSTS							
16	A		TOTLIN							
17	A		ACTMTH							
18	A		ACTYR							
19	A		STATE							
20	A		AMPAID							
21	A		K	CUST						
22	A		K	INVNUM						
23	A									
24	A									
25	A									
26	A									
27	A									
28	A									
29	A									
30	A									
31	A									
32	A									
33	A									
34	A									
35	A									
36	A									
37	A									
38	A									
39	A									
40	A									
41	A									
42	A									
43	A									
44	A									
45	A									
46	A									
47	A									
48	A									
49	A									
50	A									
51	A									
52	A									
53	A									
54	A									
55	A									
56	A									
57	A									
58	A									
59	A									
60	A									
61	A									
62	A									
63	A									
64	A									
65	A									
66	A									
67	A									
68	A									
69	A									
70	A									
71	A									
72	A									
73	A									
74	A									
75	A									
76	A									
77	A									
78	A									
79	A									
80	A									

*Number of sheets per pad may vary slightly.

Figure 7-13 (Part 1 of 17). Sample Payment Update Program

File	Keying Instruction	Graphic Key							
Programmer	Date								

Description	Page	of
-------------	------	----

Sequence Number	Form Type And/Or Comment (A/O/*)	Conditioning				Name	Length	Reference (R)	Data Type (B A/P/S/B A/S/X/Y/N/I/M)	Location		Functions
		Indicator	Not (N)	Indicator	Not (N)					Line	Pos	
A*					DDS FOR THE DISPLAY DEVICE FILE							ARCØ1ØØ
A*					ACCOUNTS RECEIVABLE INTERACTIVE PAYMENT UPDATE							
A*					R SUBFILE 1							SFL
A												TEXT('Subfile for customer payment')
A*					ACPPMT	4A			I	5		TEXT('Accept payment')
A												VALUES('*YES' '*NO')
A		51										DSPATR(RI MDT)
A		N51										DSPATR(ND PR)
A*					CUST	5			B	5		TEXT('Customer number')
A		52										DSPATR(RI)
A		53										DSPATR(ND)
A		54										DSPATR(PR)
A*					AMPAID	8			Ø2B	5		TEXT('Amount paid')
A												CHECK(FE)
A												AUTO(RAB)
A												CMP(GT Ø)
A		52										DSPATR(RI)
A		53										DSPATR(ND)
A		54										DSPATR(PR)
A*					ECPMSG	31A			Ø	5		TEXT('Exception message')
A		52										DSPATR(RI)
A		53										DSPATR(ND)
A		54										DSPATR(BL)
A*					OVRPMT	8Y			20	5		TEXT('Over payment')
A												EDTCDE(1)
A		55										DSPATR(BL)
A		N56										DSPATR(ND)
A*					STSCDE	1A			H			TEXT('Status code')

*Number of sheets per pad may vary slightly.

Figure 7-13 (Part 2 of 17). Sample Payment Update Program

COBOL SOURCE LISTING

```

STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN 5 COPYNAME CHG/DAT

 1 000100 IDENTIFICATION DIVISION.
 2 000200 PROGRAM-ID.
 3 000300 ARCO10.
 4 000400 ENVIRONMENT DIVISION.
 5 000500 CONFIGURATION SECTION.
 6 000600 SOURCE-COMPUTER.
 7 000700 IBM-S38.
 8 000800 OBJECT-COMPUTER.
 9 000900 IBM-S38.
10 001000 INPUT-OUTPUT SECTION.
11 001100 FILE-CONTROL.
12 001200 SELECT CUSTOMER-INVOICE-FILE
13 001300 ASSIGN TO DATABASE-ORDHDL
14 001400 ORGANIZATION IS INDEXED
15 001500 ACCESS MODE IS SEQUENTIAL
16 001600 RECORD KEY IS COMP-KEY
17 001700 FILE STATUS IS STATUS-CODE-ONE.
18 001800 SELECT CUSTOMER-MASTER-FILE
19 001900 ASSIGN TO DATABASE-CUSMSTP
20 002000 ORGANIZATION IS INDEXED
21 002100 ACCESS MODE IS RANDOM
22 002200 RECORD KEY IS CUST OF CUSTOMER-MASTER-RECORD.
23 002300 SELECT PAYMENT-UPDATE-DISPLAY-FILE
24 002400 ASSIGN TO WORKSTATION-ARCO10D
25 002500 ORGANIZATION IS TRANSACTION
26 002600 ACCESS MODE IS DYNAMIC
27 002700 RELATIVE KEY IS REL-NUMBER
28 002800 FILE STATUS IS STATUS-CODE-ONE
29 002900 CONTROL-AREA IS WS-CONTROL.
30 003000 DATA DIVISION.
31 003100 FILE SECTION.
32 003200 FD CUSTOMER-INVOICE-FILE
33 003300 LABEL RECORDS ARE STANDARD.
34 003400 U1 CUSTOMER-INVOICE-RECORD.
35 003500 COPY DUS-ORDHDR OF ORDHDL.
37 *000001 ORDHDR
*000002* I-U FORMAT:ORDHDR FROM FILE ORDHDL OF LIBRARY CBL2849 ORDHDR
*000003* ORDHDR
*000004*THE KEY DEFINITIONS FOR RECORD FORMAT ORDHDR ORDHDR
*000005* NUMBER NAME RETRIEVAL TYPE ALTSEQ ORDHDR
*000006* 0001 CUST ASCENDING AN NO ORDHDR
*000007* 0002 INVNUM ASCENDING N NO ORDHDR
38 *000008 05 ORDHDR. ORDHDR
39 *000009 06 CUST PIC X(5). ORDHDR
*000010* CUSTOMER NUMBER ORDHDR
40 *000011 06 INVNUM PIC S9(5) COMP-3. ORDHDR
*000012* INVOICE NUMBER ORDHDR
41 *000013 06 ORDER PIC S9(5) COMP-3. ORDHDR
*000014* ORDER NUMBER ORDHDR
42 *000015 06 ORDDAT PIC S9(6) COMP-3. ORDHDR
*000016* DATE ORDER WAS ENTERED ORDHDR
43 *000017 06 CUSORJ PIC X(15). ORDHDR
*000018* CUSTOMER PURCHASE ORDER NUMBER ORDHDR
44 *000019 06 SHPVIA PIC X(15). ORDHDR
*000020* SHIPPING INSTRUCTIONS ORDHDR

```

Figure 7-13 (Part 4 of 17). Sample Payment Update Program

```

                                COBOL SOURCE LISTING                                ARCO10
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S COPYNAME  CHG/DATE
45 +000021          06 ORDSTS          PIC S9(1)          COMP-3.          ORDHDR
+000022*          ORDER STATUS 1PCS 2CNT 3CHK 4ROY 5PRT 6PCK  ORDHDR
46 +000023          06 OPRNAM          PIC X(10).        ORDHDR
+000024*          OPERATOR NAME WHO ENTERED THE ORDER  ORDHDR
47 +000025          06 ORDAMT          PIC S9(6)V9(2)    COMP-3.          ORDHDR
+000026*          TOTAL DOLLAR AMOUNT OF THE ORDER  ORDHDR
48 +000027          06 CUSTYP          PIC S9(1)          COMP-3.          ORDHDR
+000028*          CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=DT  ORDHDR
49 +000029          06 PRTOAT          PIC S9(6)          COMP-3.          ORDHDR
+000030*          DATE ORDER WAS PRINTED  ORDHDR
50 +000031          06 OPNSTS          PIC S9(1)          COMP-3.          ORDHDR
+000032*          ORDER OPEN STATUS 1=OPEN 2=CLOSE 3=CANCEL  ORDHDR
51 +000033          06 TOTLIN          PIC S9(3)          COMP-3.          ORDHDR
+000034*          TOTAL LINE ITEMS IN ORDER  ORDHDR
52 +000035          06 ACTMTH          PIC S9(2)          COMP-3.          ORDHDR
+000036*          ACCOUNTING MONTH OF SALE  ORDHDR
53 +000037          06 ACTYR          PIC S9(2)          COMP-3.          ORDHDR
+000038*          ACCOUNTING YEAR OF SALE  ORDHDR
54 +000039          06 STATE          PIC X(2).         ORDHDR
+000040*          STATE  ORDHDR
55 +000041          06 AMPAID          PIC S9(6)V9(2)    COMP-3.          ORDHDR
+000042*          TOTAL DOLLAR AMOUNT PAID  ORDHDR
56 003600 65  COMP-KEY RENAMES CUST THRU INVNUM.
57 003700 FD  CUSTOMER-MASTER-FILE
58 003800 LABEL RECORDS ARE STANDARD.
59 003900 01  CUSTOMER-MASTER-RECORD.
60 004000 COPY DDS-CUSMST OF CUSMSTP.
62 +000001          I-O FORMAT:CUSMST FROM FILE CUSMSTP OF LIBRARY CBL2849  CUSMST
+000002*          CUSTOMER MASTER RECORD  CUSMST
+000003*          THE KEY DEFINITIONS FOR RECORD FORMAT CUSMST  CUSMST
+000004*          NUMBER NAME RETRIEVAL TYPE ALTSEQ  CUSMST
+000005*          0001 CUST ASCENDING AN NO  CUSMST
+000006*          05 CUSMST.  CUSMST
63 +000007          06 CUST          PIC X(5).         CUSMST
+000008          CUSTOMER NUMBER  CUSMST
+000009*          06 NAME          PIC X(25).        CUSMST
+000010          CUSTOMER NAME  CUSMST
66 +000012          06 ADDR          PIC X(20).        CUSMST
+000013*          CUSTOMER ADDRESS  CUSMST
67 +000014          06 CITY          PIC X(20).        CUSMST
+000015*          CUSTOMER CITY  CUSMST
68 +000016          06 STATE          PIC X(2).         CUSMST
+000017*          STATE  CUSMST
69 +000018          06 ZIP          PIC S9(5)          COMP-3.          CUSMST
+000019*          ZIP CODE  CUSMST
70 +000020          06 SRHCOU          PIC X(6).         CUSMST
+000021*          CUSTOMER NUMBER SEARCH CODE  CUSMST
71 +000022          06 CUSTYP          PIC S9(1)          COMP-3.          CUSMST
+000023*          CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=DT  CUSMST
72 +000024          06 ARBAL          PIC S9(6)V9(2)    COMP-3.          CUSMST
+000025*          ACCTS REC BALANCE  CUSMST
73 +000026          06 ORDBAL          PIC S9(6)V9(2)    COMP-3.          CUSMST
+000027*          A/R AMT IN ORDER FILE  CUSMST
74 +000028          06 LSTAMT          PIC S9(6)V9(2)    COMP-3.          CUSMST

```

Figure 7-13 (Part 5 of 17). Sample Payment Update Program

COBOL SOURCE LISTING		ARCO10	
STMT	SEQNBR -A 1 B...	2 ... 3 ... 4 ... 5 ... 6 ... 7	.IDENTFCN 5 COPYNAME CHG/DAT
+000029*		LAST AMOUNT PAID IN A/R	CUSMST
75 +000030	06 LSTDAT	PIC S9(6) COMP-3.	CUSMST
+000031*		LAST DATE PAID IN A/R	CUSMST
76 +000032	06 CRDLMT	PIC S9(6)V9(2) COMP-3.	CUSMST
+000033*		CUSTOMER CREDIT LIMIT	CUSMST
77 +000034	06 SLSYR	PIC S9(8)V9(2) COMP-3.	CUSMST
+000035*		CUSTOMER SALES THIS YEAR	CUSMST
78 +000036	06 SLSLYR	PIC S9(8)V9(2) COMP-3.	CUSMST
+000037*		CUSTOMER SALES LAST YEAR	CUSMST
79 004100 FD	PAYMENT-UPDATE-DISPLAY-FILE		
80 004200	LABEL RECORDS ARE OMITTED.		
81 004300 01	PAYMENT-UPDATE-DISPLAY-RECORD.		
82 004400	COPY DUS-ALL-FORMATS OF ARCO10D.		
84 +000001	05	ARCO10D-RECORD PIC X(59).	<-ALL-FMTS
85 +000002			<-ALL-FMTS
+000003*	INPUT FORMAT:SUBFILE1	FROM FILE ARCO10D OF LIBRARY CBL2849	<-ALL-FMTS
+000004*		SUBFILE FOR CUSTOMER PAYMENT	<-ALL-FMTS
86 +000005	05 SUBFILE1-I	REDEFINES ARCO10D-RECORD.	<-ALL-FMTS
87 +000006	06 ACPPMT	PIC X(4).	<-ALL-FMTS
+000007*		ACCEPT PAYMENT	<-ALL-FMTS
88 +000008	06 CUST	PIC X(5).	<-ALL-FMTS
+000009*		CUSTOMER NUMBER	<-ALL-FMTS
89 +000010	06 AMPAID	PIC S9(6)V9(2).	<-ALL-FMTS
+000011*		AMOUNT PAID	<-ALL-FMTS
90 +000012	06 ECPMSG	PIC X(31).	<-ALL-FMTS
+000013*		EXCEPTION MESSAGE	<-ALL-FMTS
91 +000014	06 OVRPMT	PIC S9(6)V9(2).	<-ALL-FMTS
+000015*		OVER PAYMENT	<-ALL-FMTS
92 +000016	06 STSCDE	PIC X(1).	<-ALL-FMTS
+000017*		STATUS CODE	<-ALL-FMTS
93 +000018			<-ALL-FMTS
+000019*	OUTPUT FORMAT:SUBFILE1	FROM FILE ARCO10D OF LIBRARY CBL2849	<-ALL-FMTS
+000020*		SUBFILE FOR CUSTOMER PAYMENT	<-ALL-FMTS
94 +000021	05 SUBFILE1-O	REDEFINES ARCO10D-RECORD.	<-ALL-FMTS
95 +000022	06 SUBFILE1-O-INDIC.		<-ALL-FMTS
96 +000023	07 IN51	PIC 1 INDIC 51.	<-ALL-FMTS
97 +000024	07 IN52	PIC 1 INDIC 52.	<-ALL-FMTS
98 +000025	07 IN53	PIC 1 INDIC 53.	<-ALL-FMTS
99 +000026	07 IN54	PIC 1 INDIC 54.	<-ALL-FMTS
100 +000027	07 IN55	PIC 1 INDIC 55.	<-ALL-FMTS
101 +000028	07 IN56	PIC 1 INDIC 56.	<-ALL-FMTS
102 +000029	06 CUST	PIC X(5).	<-ALL-FMTS
+000030*		CUSTOMER NUMBER	<-ALL-FMTS
103 +000031	06 AMPAID	PIC S9(6)V9(2).	<-ALL-FMTS
+000032*		AMOUNT PAID	<-ALL-FMTS
104 +000033	06 ECPMSG	PIC X(31).	<-ALL-FMTS
+000034*		EXCEPTION MESSAGE	<-ALL-FMTS
105 +000035	06 OVRPMT	PIC S9(6)V9(2).	<-ALL-FMTS
+000036*		OVER PAYMENT	<-ALL-FMTS
106 +000037	06 STSCDE	PIC X(1).	<-ALL-FMTS
+000038*		STATUS CODE	<-ALL-FMTS
107 +000039			<-ALL-FMTS
+000040*	INPUT FORMAT:CONTROL1	FROM FILE ARCO10D OF LIBRARY CBL2849	<-ALL-FMTS
+000041*		SUBFILE CONTROL	<-ALL-FMTS
108 +000042	05 CONTROL1-I	REDEFINES ARCO10D-RECORD.	<-ALL-FMTS

Figure 7-13 (Part 6 of 17). Sample Payment Update Program

COBOL SOURCE LISTING		ARCO10	
STMT	SEQNBR -A 1 B.. ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S	COPYNAME	CHG/DATE
109	+000043	06 CONTROL1-I-INDIC.	<-ALL-FMTS
110	+000044	07 IN99 PIC 1 INDIC 99.	<-ALL-FMTS
	+000045*	HELP KEY	<-ALL-FMTS
111	+000046	07 IN98 PIC 1 INDIC 98.	<-ALL-FMTS
	+000047*	END PAYMENT UPDATE	<-ALL-FMTS
112	+000048	07 IN97 PIC 1 INDIC 97.	<-ALL-FMTS
	+000049*	IGNORE INPUT	<-ALL-FMTS
113	+000050		<-ALL-FMTS
	+000051*	OUTPUT FORMAT:CONTROL1 FROM FILE ARCO100 OF LIBRARY CBL2849	<-ALL-FMTS
	+000052*	SUBFILE CONTROL	<-ALL-FMTS
114	+000053	05 CONTROL1-O REDEFINES ARCO100-RECORD.	<-ALL-FMTS
115	+000054	06 CONTROL1-O-INDIC.	<-ALL-FMTS
116	+000055	07 IN61 PIC 1 INDIC 61.	<-ALL-FMTS
117	+000056	07 IN62 PIC 1 INDIC 62.	<-ALL-FMTS
118	+000057	07 IN99 PIC 1 INDIC 99.	<-ALL-FMTS
	+000058*	HELP KEY	<-ALL-FMTS
119	+000059	07 IN63 PIC 1 INDIC 63.	<-ALL-FMTS
120	+000060	07 IN64 PIC 1 INDIC 64.	<-ALL-FMTS
121	+000061		<-ALL-FMTS
	+000062*	INPUT FORMAT:MESSAGE1 FROM FILE ARCO100 OF LIBRARY CBL2849	<-ALL-FMTS
	+000063*	MESSAGE RECORD	<-ALL-FMTS
	+000064*	05 MESSAGE1-I REDEFINES ARCO100-RECORD.	<-ALL-FMTS
122	+000065		<-ALL-FMTS
	+000066*	OUTPUT FORMAT:MESSAGE1 FROM FILE ARCO100 OF LIBRARY CBL2849	<-ALL-FMTS
	+000067*	MESSAGE RECORD	<-ALL-FMTS
123	+000068	05 MESSAGE1-O REDEFINES ARCO100-RECORD.	<-ALL-FMTS
124	+000069	06 MESSAGE1-O-INDIC.	<-ALL-FMTS
125	+000070	07 IN71 PIC 1 INDIC 71.	<-ALL-FMTS
126	004500	WORKING-STORAGE SECTION.	
127	004600	01 REL-NUMBER PIC 9(05)	
128	004700	VALUE ZEROS.	
129	004800	01 WS-CONTROL.	
130	004900	05 WS-INO PIC X(02).	
131	005000	05 WS-FORMAT PIC X(10).	
132	005100	01 SYSTEM-DATE.	
133	005200	05 SYSTEM-YEAR PIC 99.	
134	005300	05 SYSTEM-MONTH PIC 99.	
135	005400	05 SYSTEM-DAY PIC 99.	
136	005500	01 PROGRAM-DATE.	
137	005600	05 PROGRAM-MONTH PIC 99.	
138	005700	05 PROGRAM-DAY PIC 99.	
139	005800	05 PROGRAM-YEAR PIC 99.	
140	005900	01 FILE-DATE REDEFINES PROGRAM-DATE PIC 59(6).	
141	006000	01 EXCEPTION-STATUS.	
142	006100	05 STATUS-CODE-ONE PIC XX.	
143	006200	88 SUBFILE-IS-FULL VALUE "9M".	
144	006300	01 EXCEPTION-MESSAGES.	
145	006400	05 MESSAGE-ONE PIC X(31)	
146	006500	VALUE "CUSTOMER DOES NOT EXIST"	
147	006600	05 MESSAGE-TWO PIC X(31)	
148	006700	VALUE "NO INVOICES EXIST FOR CUSTOMER "	
149	006800	05 MESSAGE-THREE PIC X(31)	
150	006900	VALUE "CUSTOMER HAS AN OVER PAYMENT UF".	
151	007000	01 PROGRAM-VARIABLES.	
152	007100	05 AMOUNT-DWEU PIC 59(6)V99.	

Figure 7-13 (Part 7 of 17). Sample Payment Update Program

COBOL SOURCE LISTING				ARCO10
STMT	SEQNBR	-A 1 B...	2 ... 3 ... 4 ... 5 ... 6 ... 7	.IDENTFCN S COPYNAME CHG/DAT
153	007200	05	AMOUNT-PAID	PIC 59(6)V99.
154	007300	05	INVOICE-BALANCE	PIC 59(6)V99.
155	007400	01	ERRPGM-PARAMETERS.	
156	007500	05	DISPLAY-PARAMETER	PIC X(008)
157	007600			VALUE "ARCO10D "
158	007700	05	DUMMY-ONE	PIC X(006)
159	007800			VALUE SPACES.
160	007900	05	DUMMY-TWO	PIC X(008)
161	008000			VALUE SPACES.
162	008100	05	STATUS-CODE-TWO.	
163	008200	10	PRIMARY	PIC X.
164	008300	10	SECONDARY	PIC X.
165	008400	10	FILLER	PIC X(005)
166	008500			VALUE SPACES.
167	008600	05	DUMMY-THREE	PIC X(010)
168	008700			VALUE SPACES.
169	008800	01	SWITCH-AREA.	
170	008900	05	SW01	PIC 1.
171	009000	88	WRITE-DISPLAY	VALUE B'1'.
172	009100	88	READ-DISPLAY	VALUE B'0'.
173	009200	05	SW02	PIC 1.
174	009300	88	SUBFILE1-FORMAT	VALUE B'1'.
175	009400	88	NOT-SUBFILE1-FORMAT	VALUE B'0'.
176	009500	05	SW03	PIC 1.
177	009600	88	CONTROL1-FORMAT	VALUE B'1'.
178	009700	88	NOT-CONTROL1-FORMAT	VALUE B'0'.
179	009800	05	SW04	PIC 1.
180	009900	88	NO-MORE-TRANSACTIONS-EXIST	VALUE B'1'.
181	010000	88	TRANSACTIONS-EXIST	VALUE B'0'.
182	010100	05	SW05	PIC 1.
183	010200	88	CUSTOMER-NOT-FOUND	VALUE B'1'.
184	010300	88	CUSTOMER-EXIST	VALUE B'0'.
185	010400	05	SW06	PIC 1.
186	010500	88	NO-MORE-INVOICES-EXIST	VALUE B'1'.
187	010600	88	CUSTOMER-INVOICE-EXIST	VALUE B'0'.
188	010700	05	SW07	PIC 1.
189	010800	88	NO-MORE-PAYMENT-EXIST	VALUE B'1'.
190	010900	88	PAYMENT-EXIST	VALUE B'0'.
191	011000	05	SW08	PIC 1.
192	011100	88	INPUT-ERRORS-EXIST	VALUE B'1'.
193	011200	88	NO-INPUT-ERRORS-EXIST	VALUE B'0'.
194	011300	05	SW09	PIC 1.
195	011400	88	OVER-PAYMENT-DISPLAYED-ONCE	VALUE B'1'.
196	011500	88	OVER-PAYMENT-NOT-DISPLAYED	VALUE B'0'.
197	011600	01	INDICATOR-AREA.	
198	011700	05	IN99	PIC 1 INDIC 99.
199	011800	88	HELP-IS-NEEDED	VALUE B'1'.
200	011900	88	HELP-IS-NOT-NEEDED	VALUE B'0'.
201	012000	05	IN98	PIC 1 INDIC 98.
202	012100	88	END-OF-PAYMENT-UPDATE	VALUE B'1'.
203	012200	05	IN97	PIC 1 INDIC 97.
204	012300	88	IGNORE-INPUT	VALUE B'1'.
205	012400	05	IN51	PIC 1 INDIC 51.
206	012500	88	DISPLAY-ACCEPT-PAYMENT	VALUE B'1'.
207	012600	88	DO-NOT-DISPLAY-ACCEPT-PAYMENT	VALUE B'0'.

Figure 7-13 (Part 8 of 17). Sample Payment Update Program

STMT	SEQNBR	-A	1	B...	...	2	3	4	5	6	7	.IDENTFCN	S	COPYNAME	CHG/DATE
208	012700		05	INS2																					
209	012800			88	REVERSE-FIELD-IMAGE																				
210	012900			88	DO-NOT-REVERSE-FIELD-IMAGE																				
211	013000		05	INS3																					
212	013100			88	DO-NOT-DISPLAY-FIELD																				
213	013200			88	DISPLAY-FIELD																				
214	013300		05	INS4																					
215	013400			88	PROTECT-INPUT-FIELD																				
216	013500			88	DO-NOT-PROTECT-INPUT-FIELD																				
217	013600		05	INS5																					
218	013700			88	MAKE-FIELD-BLINK																				
219	013800			88	DO-NOT-MAKE-FIELD-BLINK																				
220	013900		05	INS6																					
221	014000			88	DISPLAY-OVER-PAYMENT																				
222	014100			88	DO-NOT-DISPLAY-OVER-PAYMENT																				
223	014200		05	IN61																					
224	014300			88	CLEAR-SUBFILE																				
225	014400			88	DO-NOT-CLEAR-SUBFILE																				
226	014500		05	IN62																					
227	014600			88	DISPLAY-SCREEN																				
228	014700			88	DO-NOT-DISPLAY-SCREEN																				
229	014800		05	IN63																					
230	014900			88	DISPLAY-ACCEPT-HEADING																				
231	015000			88	DO-NOT-DISPLAY-ACCEPT-HEADING																				
232	015100		05	IN64																					
233	015200			88	DISPLAY-EXCEPTION																				
234	015300			88	DO-NOT-DISPLAY-EXCEPTION																				
235	015400		05	IN71																					
236	015500			88	DISPLAY-ACCEPT-MESSAGE																				
237	015600			88	DO-NOT-DISPLAY-ACCEPT-MESSAGE																				
238	015700				PROCEDURE DIVISION.																				
	015800				DECLARATIVES.																				
	015900				TRANSACTION-ERROR SECTION.																				
	016000				USE AFTER STANDARD ERROR PROCEDURE																				
	016100				PAYMENT-UPDATE-DISPLAY-FILE.																				
	016200				WORK-STATION-ERROR-HANDLER.																				
239	016300				IF SUBFILE-IS-FULL THEN NEXT SENTENCE																				
	016400				ELSE																				
240	016500				DISPLAY 'ERROR IN PAYMENT-UPDATE' STATUS-CODE-ONE.																				
	016600				END DECLARATIVES.																				
	016700				CUSTOMER-PAYMENT-UPDATE SECTION.																				
	016800				MAINLINE-ROUTINE.																				
241	016900				PERFORM SET-UP-ROUTINE.																				
242	017000				PERFORM PROCESS-TRANSACTION-FILE																				
	017100				UNTIL END-OF-PAYMENT-UPDATE.																				
243	017200				PERFORM CLEAN-UP-ROUTINE.																				
	017300				SET-UP-ROUTINE.																				
244	017400				OPEN I-O CUSTOMER-INVOICE-FILE																				
	017500				CUSTOMER-MASTER-FILE																				
	017600				PAYMENT-UPDATE-DISPLAY-FILE.																				
245	017700				MOVE B'0' TO INDICATOR-AREA																				
	017800				SWITCH-AREA.																				
246	017900				ACCEPT SYSTEM-DATE FROM DATE.																				
247	018000				MOVE SYSTEM-YEAR TO PROGRAM-YEAR.																				
248	018100				MOVE SYSTEM-MONTH TO PROGRAM-MONTH.																				

Figure 7-13 (Part 9 of 17). Sample Payment Update Program

```

STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME CHG/DA

249 018200     MOVE SYSTEM-DAY TO PROGRAM-DAY.
250 018300     SET WRITE-DISPLAY, CONTROL1-FORMAT,
018400         DO-NOT-DISPLAY-OVER-PAYMENT,
018500         DO-NOT-PROTECT-INPUT-FIELD,
018600         DO-NOT-REVERSE-FIELD-IMAGE,
018700         DO-NOT-MAKE-FIELD-BLINK,
018800         DO-NOT-DISPLAY-ACCEPT-PAYMENT,
018900         CLEAR-SUBFILE TO TRUE.
251 019000     MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC.
252 019100     WRITE PAYMENT-UPDATE-DISPLAY-RECORD
019200         FORMAT IS 'CONTROL1'.
253 019300     SET DO-NOT-CLEAR-SUBFILE TO TRUE.
254 019400     PERFORM INITIALIZE-SUBFILE-RECORD 17 TIMES.
255 019500     SET DISPLAY-SCREEN TO TRUE.
256 019600     MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC.
257 019700     WRITE PAYMENT-UPDATE-DISPLAY-RECORD
019800         FORMAT IS 'CONTROL1'.
258 019900     READ PAYMENT-UPDATE-DISPLAY-FILE RECORD
020000         FORMAT IS 'CONTROL1'.
259 020100     MOVE CORR CONTROL1-I-INDIC TO INDICATOR-AREA.
020200     PROCESS-TRANSACTION-FILE.
260 020300     IF HELP-IS-NOT-NEEDED THEN
261 020400         IF IGNORE-INPUT THEN
262 020500             SET WRITE-DISPLAY, CONTROL1-FORMAT
020600                 CLEAR-SUBFILE, DISPLAY-FIELD
020700                 DO-NOT-DISPLAY-OVER-PAYMENT
020800                 DO-NOT-PROTECT-INPUT-FIELD
020900                 DO-NOT-REVERSE-FIELD-IMAGE
021000                 DO-NOT-DISPLAY-ACCEPT-PAYMENT
021100                 DO-NOT-DISPLAY-ACCEPT-HEADING
021200                 DO-NOT-DISPLAY-ACCEPT-MESSAGE
021300                 DO-NOT-DISPLAY-EXCEPTION
021400                 DO-NOT-MAKE-FIELD-BLINK TO TRUE
263 021500         MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC
264 021600         WRITE PAYMENT-UPDATE-DISPLAY-RECORD
021700             FORMAT IS 'CONTROL1'
265 021800         SET DO-NOT-CLEAR-SUBFILE TO TRUE
266 021900         MOVE 0 TO REL-NUMBER
267 022000         PERFORM INITIALIZE-SUBFILE-RECORD 17 TIMES
022100     ELSE
268 022200         SET TRANSACTIONS-EXIST
022300             DO-NOT-DISPLAY-ACCEPT-HEADING
022400             DO-NOT-DISPLAY-ACCEPT-MESSAGE
022500             DO-NOT-DISPLAY-EXCEPTION TO TRUE
269 022600         PERFORM READ-MODIFIED-SUBFILE-RECORD
270 022700         PERFORM TRANSACTION-VALIDATION
022800             UNTIL NO-MORE-TRANSACTIONS-EXIST
271 022900         SET NO-INPUT-ERRORS-EXIST TO TRUE
272 023000         PERFORM TEST-FOR-RECORD-INPUT-ERRORS
023100             VARYING REL-NUMBER
023200                 FROM 1
023300                 BY 1
023400             UNTIL REL-NUMBER IS GREATER THAN 17
023500             OR INPUT-ERRORS-EXIST
273 023600         IF NO-INPUT-ERRORS-EXIST THEN

```

Figure 7-13 (Part 10 of 17). Sample Payment Update Program

```

                                COBOL SOURCE LISTING                                ARC010
STMT SEQNBR -A 1 8... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S  COPYNAME  CHG/DATE
274 023700          IF OVER-PAYMENT-DISPLAYED-ONCE THEN
275 023800          SET WRITE-DISPLAY, CONTROL1-FORMAT
023900              DO-NOT-DISPLAY-OVER-PAYMENT
024000              DO-NOT-PROTECT-INPUT-FIELD
024100              DO-NOT-REVERSE-FIELD-IMAGE
024200              DO-NOT-MAKE-FIELD-BLINK
024300              DO-NOT-DISPLAY-ACCEPT-PAYMENT
024400              DO-NOT-DISPLAY-ACCEPT-HEADING
024500              DO-NOT-DISPLAY-ACCEPT-MESSAGE
024600              DO-NOT-DISPLAY-EXCEPTION
024700              CLEAR-SUBFILE, DISPLAY-FIELD TO TRUE
276 024800          MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC
277 024900          WRITE PAYMENT-UPDATE-DISPLAY-RECORD
025000              FORMAT IS 'CONTROL1'
278 025100          SET DO-NOT-CLEAR-SUBFILE TO TRUE
279 025200          MOVE 0 TO REL-NUMBER
280 025300          PERFORM INITIALIZE-SUBFILE-RECORD 17 TIMES
025400          ELSE
281 025500              SET OVER-PAYMENT-DISPLAYED-ONCE TO TRUE
025600          ELSE
025700              NEXT SENTENCE
025800          ELSE
025900              NEXT SENTENCE.
282 026000          SET WRITE-DISPLAY, DISPLAY-SCREEN TO TRUE.
283 026100          MOVE CORR INDICATOR-AREA TO MESSAGE1-O-INDIC.
284 026200          WRITE PAYMENT-UPDATE-DISPLAY-RECORD
026300              FORMAT IS 'MESSAGE1'.
285 026400          SET WRITE-DISPLAY, CONTROL1-FORMAT TO TRUE.
286 026500          MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC.
287 026600          WRITE PAYMENT-UPDATE-DISPLAY-RECORD
026700              FORMAT IS 'CONTROL1'.
288 026800          READ PAYMENT-UPDATE-DISPLAY-FILE RECORD
026900              FORMAT IS 'CONTROL1'.
289 027000          MOVE CORR CONTROL1-I-INDIC TO INDICATOR-AREA.
027100          READ-MODIFIED-SUBFILE-RECORD.
290 027200          READ SUBFILE PAYMENT-UPDATE-DISPLAY-FILE
027300              NEXT MODIFIED RECORD FORMAT IS 'SUBFILE1'
027400              AT END SET NO-MORE-TRANSACTIONS-EXIST TO TRUE.
291 027500          TEST-FUR-RECORD-INPUT-ERRORS.
292 027600          READ SUBFILE PAYMENT-UPDATE-DISPLAY-FILE RECORD
027700              FORMAT IS 'SUBFILE1'.
293 027800          IF STSCODE OF SUBFILE1-I IS EQUAL TO '1' THEN
294 027900              SET INPUT-ERRORS-EXIST TO TRUE
028000          ELSE
028100              NEXT SENTENCE.
028200          TRANSACTION-VALIDATION.
295 028300          MOVE CUST OF SUBFILE1-I OF PAYMENT-UPDATE-DISPLAY-RECORD
028400              TO CUST OF CUSTOMER-MASTER-RECORD.
296 028500          SET CUSTOMER-EXIST TO TRUE.
297 028600          READ CUSTOMER-MASTER-FILE
028700              KEY IS CUST OF CUSTOMER-MASTER-RECORD
298 028800              INVALID KEY SET CUSTOMER-NOT-FOUND TO TRUE.
299 028900          IF CUSTOMER-EXIST THEN
300 029000              MOVE CUST OF CUSMST TO CUST OF ORDHOR
301 029100              MOVE ZEROES TO INVNUM

```

Figure 7-13 (Part 11 of 17). Sample Payment Update Program

```

                                COBOL SOURCE LISTING                                ARCO10
STMT SEUNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S  COPYNAME  CHG/DA*

302 029200      SET CUSTOMER-INVOICE-EXIST TO TRUE
303 029300      PERFORM START-ON-CUSTOMER-INVOICE-FILE
304 029400      IF CUSTOMER-INVOICE-EXIST THEN
305 029500          PERFORM READ-CUSTOMER-INVOICE-RECORD
306 029600          IF CUSTOMER-INVOICE-EXIST THEN
307 029700              PERFORM CUSTOMER-MASTER-FILE-UPDATE
308 029800              MOVE AMPAID OF SUBFILE1-1 TO AMOUNT-PAID
309 029900              SET PAYMENT-EXIST TO TRUE
310 030000              PERFORM PAYMENT-UPDATE
310 030100                  UNTIL NO-MORE-INVOICES-EXIST
310 030200                  OR NO-MORE-PAYMENT-EXIST
311 030300          IF ARBAL OF CUSTOMER-MASTER-RECORD IS NEGATIVE
312 030400              SET MAKE-FIELD-BLINK, DISPLAY-FIELD,
312 030500                  DO-NOT-REVERSE-FIELD-IMAGE
312 030600                  OVER-PAYMENT-NOT-DISPLAYED
312 030700                  DISPLAY-OVER-PAYMENT
312 030800                  DISPLAY-EXCEPTION
312 030900                  DO-NOT-DISPLAY-ACCEPT-PAYMENT
312 031000                  PROTECT-INPUT-FIELD TO TRUE
313 031100              MOVE ARBAL TO OVRPMT OF SUBFILE1-0
314 031200              MOVE MESSAGE-THREE TO ECPMSG OF SUBFILE1-0
315 031300              MOVE '0' TO STSCDE OF SUBFILE1-0
316 031400              PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
316 031500          ELSE
317 031600              SET DO-NOT-DISPLAY-FIELD
317 031700                  DO-NOT-DISPLAY-OVER-PAYMENT
317 031800                  DO-NOT-REVERSE-FIELD-IMAGE
317 031900                  DO-NOT-MAKE-FIELD-BLINK
317 032000                  DO-NOT-DISPLAY-ACCEPT-PAYMENT
317 032100                  PROTECT-INPUT-FIELD TO TRUE
318 032200              MOVE SPACES TO ECPMSG OF SUBFILE1-0
319 032300              MOVE ZEROES TO OVRPMT OF SUBFILE1-0
320 032400              MOVE '0' TO STSCDE OF SUBFILE1-0
321 032500              PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
321 032600          ELSE
322 032700              PERFORM NO-CUSTOMER-INVOICE-ROUTINE
322 032800          ELSE
323 032900              PERFORM NO-CUSTOMER-INVOICE-ROUTINE
323 033000          ELSE
324 033100              SET REVERSE-FIELD-IMAGE
324 033200                  DO-NOT-PROTECT-INPUT-FIELD, DISPLAY-FIELD
324 033300                  DO-NOT-DISPLAY-OVER-PAYMENT
324 033400                  DO-NOT-MAKE-FIELD-BLINK
324 033500                  DISPLAY-EXCEPTION
324 033600                  DO-NOT-DISPLAY-ACCEPT-PAYMENT
324 033700                  DO-NOT-PROTECT-INPUT-FIELD TO TRUE
325 033800              MOVE ZEROES TO OVRPMT OF SUBFILE1-0
326 033900              MOVE MESSAGE-ONE TO ECPMSG OF SUBFILE1-0
327 034000              MOVE '1' TO STSCDE OF SUBFILE1-0
328 034100              PERFORM REWRITE-DISPLAY-SUBFILE-RECORD.
329 034200          PERFORM READ-MODIFIED-SUBFILE-RECORD.
329 034300          START-ON-CUSTOMER-INVOICE-FILE.
330 034400          START CUSTOMER-INVOICE-FILE
330 034500              KEY IS GREATER THAN COMP-KEY
331 034600          INVALID KEY SET NO-MORE-INVOICES-EXIST TO TRUE.

```

Figure 7-13 (Part 12 of 17). Sample Payment Update Program

```

                                COBOL SOURCE LISTING                                ARCO10
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S COPYNAME CHG/DATE

034700 READ-CUSTOMER-INVOICE-RECORD.
332 034800     READ CUSTOMER-INVOICE-FILE NEXT RECORD
333 034900     AT END SET NO-MORE-INVOICES-EXIST TO TRUE.
334 035000     IF CUST OF CUSTOMER-MASTER-RECORD
035100         IS NOT EQUAL TO CUST OF CUSTOMER-INVOICE-RECORD THEN
335 035200         SET NO-MORE-INVOICES-EXIST TO TRUE
035300     ELSE
035400         NEXT SENTENCE.
035500 CUSTOMER-MASTER-FILE-UPDATE.
336 035600     MOVE FILE-DATE TO LSTOAT OF CUSTOMER-MASTER-RECORD.
337 035700     MOVE AMPAID OF SUBFILE1-I
035800         TO LSTAMT OF CUSTOMER-MASTER-RECORD.
338 035900     SUBTRACT AMPAID OF SUBFILE1-I
036000         FROM ARBAL OF CUSTOMER-MASTER-RECORD.
339 036100     REWRITE CUSTOMER-MASTER-RECORD.
036200     REWRITE-DISPLAY-SUBFILE-RECORD.
340 036300     MOVE AMPAID OF SUBFILE1-I TO AMPAID OF SUBFILE1-O.
341 036400     MOVE CUST OF SUBFILE1-I TO CUST OF SUBFILE1-O.
342 036500     SET WRITL-DISPLAY, SUBFILE1-FORMAT TO TRUE.
343 036600     MOVE CORR INDICATOR-AREA TO SUBFILE1-O-INDIC.
344 036700     REWRITE SUBFILE PAYMENT-UPDATE-DISPLAY-RECORD
036800         FORMAT IS 'SUBFILE1'.
036900 NO-CUSTOMER-INVOICE-ROUTINE.
345 037000     IF STSCDE OF SUBFILE1-I IS EQUAL TO '1' THEN
346 037100         IF ACPMPT OF SUBFILE1-I IS EQUAL TO '*NO' THEN
347 037200             SET DO-NOT-DISPLAY-FIELD
037300                 DO-NOT-DISPLAY-OVER-PAYMENT
037400                 DO-NOT-REVERSE-FIELD-IMAGE
037500                 DO-NOT-MAKE-FIELD-BLINK
037600                 DO-NOT-DISPLAY-ACCEPT-PAYMENT
037700                 PROTECT-INPUT-FIELD TO TRUE
348 037800             MOVE SPACES TO ECPMSG OF SUBFILE1-O
349 037900             MOVE ZEROS TO OVRPMT OF SUBFILE1-O
350 038000             MOVE '0' TO STSCDE OF SUBFILE1-O
351 038100             PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
038200         ELSE
352 038300             PERFORM CUSTOMER-MASTER-FILE-UPDATE
353 038400             SET MAKE-FIELD-BLINK, DISPLAY-FIELD,
038500                 DO-NOT-REVERSE-FIELD-IMAGE
038600                 OVER-PAYMENT-NOT-DISPLAYED
038700                 DISPLAY-OVER-PAYMENT
038800                 DISPLAY-EXCEPTION
038900                 DO-NOT-DISPLAY-ACCEPT-PAYMENT
039000                 PROTECT-INPUT-FIELD TO TRUE
354 039100             MOVE ARBAL TO OVRPMT OF SUBFILE1-O
355 039200             MOVE MESSAGE-THREE TO ECPMSG OF SUBFILE1-O
356 039300             MOVE '0' TO STSCDE OF SUBFILE1-O
357 039400             PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
039500         ELSE
358 039600             SET REVERSE-FIELD-IMAGE, DISPLAY-FIELD
039700                 DO-NOT-PROTECT-INPUT-FIELD
039800                 DO-NOT-DISPLAY-OVER-PAYMENT
039900                 DISPLAY-EXCEPTION
040000                 DISPLAY-ACCEPT-PAYMENT
040100                 DISPLAY-ACCEPT-HEADING

```

Figure 7-13 (Part 13 of 17). Sample Payment Update Program

STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME CHG/DAT

```

040200      DISPLAY-ACCEPT-MESSAGE
040300      DO-NOT-MAKE-FIELD-BLINK TO TRUE
359 040400      MOVE ZEROES TO DVRPMT OF SUBFILE1-0
360 040500      MOVE MESSAGE-TWO TO ECPMSG OF SUBFILE1-0
361 040600      MOVE '1' TO STSCDE OF SUBFILE1-0
362 040700      PERFORM REWRITL-DISPLAY-SUBFILE-RECORD.
040800      PAYMENT-UPDATE.
363 040900      SUBTRACT AMPAID OF CUSTOMER-INVOICE-RECORD
041000      FROM ORDAMT OF CUSTOMER-INVOICE-RECORD
041100      GIVING AMOUNT-OWED.
364 041200      SUBTRACT AMOUNT-PAID
041300      FROM AMOUNT-OWED
041400      GIVING INVOICE-BALANCE.
365 041500      IF INVOICE-BALANCE IS LESS THAN .01 THEN
366 041600          MOVE 2 TO UPNSTS OF CUSTOMER-INVOICE-RECORD
367 041700          MOVE ORDAMT OF CUSTOMER-INVOICE-RECORD
041800          TO AMPAID OF CUSTOMER-INVOICE-RECORD
368 041900          SUBTRACT AMOUNT-OWED FROM AMOUNT-PAID
042000      ELSE
369 042100          ADD AMOUNT-PAID TO AMPAID OF CUSTOMER-INVOICE-RECORD
370 042200          SET NO-MORE-PAYMENT-EXIST TO TRUE.
371 042300      REWRITE CUSTOMER-INVOICE-RECORD.
372 042400      IF NO-MORE-PAYMENT-EXIST THEN
042500          NEXT SENTENCE
042600      ELSE
373 042700          PERFORM READ-CUSTOMER-INVOICE-RECORD.
042800      INITIALIZE-SUBFILE-RECORD.
374 042900      MOVE SPACES TO CUST OF SUBFILE1-0
043000          ECPMSG OF SUBFILE1-0.
375 043100      MOVE '0' TO STSCDE OF SUBFILE1-0.
376 043200      MOVE ZEROES TO AMPAID OF SUBFILE1-0
043300          OVRPMT OF SUBFILE1-0.
377 043400      ADD 1 TO REL-NUMBER.
378 043500      MOVE CURR INDICATOR-AREA TO SUBFILE1-0-INDIC.
379 043600      WRITE SUBFILE PAYMENT-UPDATE-DISPLAY-RECORD
043700          FORMAT IS 'SUBFILE1'.
043800      CLEAN-UP-ROUTINE.
380 043900      CLOSE CUSTOMER-INVOICE-FILE
044000          CUSTOMER-MASTER-FILE
044100          PAYMENT-UPDATE-DISPLAY-FILE.
381 044200      STOP RUN.

```

***** END OF SOURCE *****

Figure 7-13 (Part 14 of 17). Sample Payment Update Program

This is the initial display that is written to the work station to prompt the user to enter the customer number and payment:

Customer Payment Update Prompt		Date 11/10/80
Customer	Payment	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	

The user enters the customer numbers and payments:

Customer Payment Update Prompt		Date 11/10/80
Customer	Payment	
34500	2000	
40500	30000	
36000	2500	
12500	200	
22799	4500	
41900	7500	
10001	5000	
49500	2500	
13300	3500	
56900	4000	

Figure 7-13 (Part 15 of 17). Sample Payment Update Program

Payments that would result in overpayments or that have invalid customer numbers are left on the display and appropriate messages are added:

Customer Payment Update Prompt			Date 11/10/80
Accept Payment	Customer	Payment	Exception message
—	40500	30000	NO INVOICES EXIST FOR CUSTOMER
—	12500	200	NO INVOICES EXIST FOR CUSTOMER
—	41900 10001	7500 5000	NO INVOICES EXIST FOR CUSTOMER CUSTOMER DOES NOT EXIST
—	13300	3500	NO INVOICES EXIST FOR CUSTOMER

Accept payment values: (*NO *YES)

The user indicates which payments to accept:

Customer Payment Update Prompt			Date 11/10/80
Accept Payment	Customer	Payment	Exception message
*NO	40500	30000	NO INVOICES EXIST FOR CUSTOMER
*YES	12500	200	NO INVOICES EXIST FOR CUSTOMER
*NO	41900 10001	7500 5000	NO INVOICES EXIST FOR CUSTOMER CUSTOMER DOES NOT EXIST
*NO	13300	3500	NO INVOICES EXIST FOR CUSTOMER

Accept payment values: (*NO *YES)

Figure 7-13 (Part 16 of 17). Sample Payment Update Program

The accepted payments are processed and overpayment information is displayed:

Customer Payment Update Prompt				Date
Accept Payment	Customer	Payment	Exception message	11/10/80
	12500	200	CUSTOMER HAS AN OVER PAYMENT OF	58.50
	10001	5000	CUSTOMER DOES NOT EXIST	

Figure 7-13 (Part 17 of 17). Sample Payment Update Program

_____ End of IBM Extension _____



Chapter 8. Creating and Executing Programs

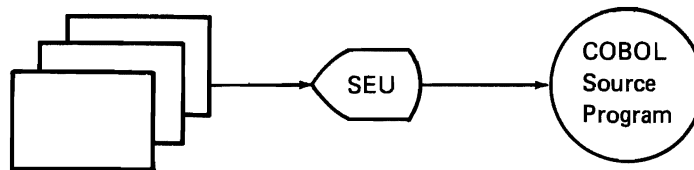
Entering the Source Program into the System

After you have written your COBOL program on the coding forms, you must enter the source program into source files in the system.

These source files should have a record length of 92. However, the COBOL compiler also supports a record length of 102. In addition to the usual fields of sequence number (6 characters), last modified date (6 characters), and the data (80 characters), a field of 10 characters that can contain additional information is placed at the end of the record (positions 93-102). This information is not used by the COBOL compiler, but is placed on the extreme right of the compiler listing. You are responsible for placing information into this field. If you want to use the additional field, create a source file with a record length of 102. The source file QCBLSRC has a record length of 92.

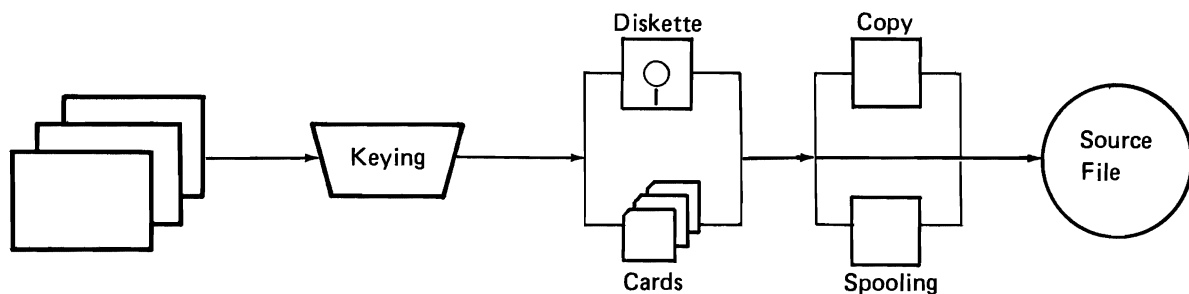
The normal ways of entering a source program are:

- You can enter it interactively by using the source entry utility (SEU) of the Interactive Data Base Utilities Licensed Program (IDU):



You can enter the EDTSRC (Edit Source) command to call SEU. Refer to the *SEU Reference Manual and User's Guide* for further information on the EDTSRC command.

- You can enter your source program in a batch manner (for example, from cards or diskettes) by using either the CPF copy or spooling functions:



Refer to the *CPF Programmer's Guide* for more information on how to use the copy or spooling function for batch entry of the source program.

Using SEU to Enter Source

SEU provides special display screen formats for COBOL that correspond to the COBOL Coding Form to help you enter your COBOL source programs. Figure 8-1 shows a display screen format, the relationship between the headings on the COBOL Coding Form and the labels on the display screen, and where you can enter the source on the display screen.

If you specify the TYPE(*CBL) parameter on the EDTSRC command, SEU invokes a COBOL syntax checker that checks each line for errors as you enter it. The COBOL syntax checker finds invalid entries in the source statements and displays error messages that allow you to correct the errors before compiling the program. Since the COBOL syntax checker checks only one statement at a time, independently of statements that precede or follow it, only syntax errors within the source data can be detected. No interrelational errors, such as undefined names and invalid references to names, are detected. These errors are detected by the COBOL compiler when the program is compiled.

Syntax checking for COBOL source functions is subject to the following rules.

- Source on a line with an asterisk (*) or a slash (/) in column 7 is not syntax checked. An asterisk indicates a comment line, a slash indicates a comment line and page eject.
- Syntax checking occurs line by line as source is entered. The error messages that are generated by lines consisting of incomplete statements disappear when the statements are completed. In the example:

```
ADD A
TO BCD.
```

An error message is generated after the first line is entered and disappears after the second line is entered, when the statement is completed. A sentence can span at most 45 lines.

IBM	COBOL Coding Form																							
SYSTEM				PUNCHING INSTRUCTIONS						PAGE OF														
PROGRAM				GRAPHIC						CARD FORM #														
PROGRAMMER				DATE	PUNCH																			
SEQUENCE	CONT	A	B	COBOL STATEMENT										IDENTIFICATION										
(PAGE)	SERIAL	1	2	4	6	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80
0	1																							
0	2																							
0	3																							
0	4																							

Area A
Area B

Indicator Area

SEU can display format lines to help you make changes or key in entries position by position.

You can key in an entry field by field in this area.

```

EDIT    US W:7   Mbr: TESTPRT                Scan: _____
FMT CB .....-A+++B+++++Pgm-i
0007.00 ENVIRONMENT DIVISION.
0008.00 CONFIGURATION SECTION.
0009.00 SOURCE-COMPUTER. IBM-S38.
0010.00 INPUT-OUTPUT SECTION.
0011.00 FILE-CONTROL.
FMT CB .....-A+++B+++++Pgm-i
0012.00 SELECT FILE-1 ASSIGN TO DATABASE-MASTER.
0013.00 SELECT FILE-2 ASSIGN TO DATABASE-MASTER.
0014.00 DATA DIVISION.
0015.00 FILE SECTION.
0016.00 FD FILE-1
0017.00 LABEL RECORDS ARE STANDARD
0018.00 RECORD CONTAINS 20 CHARACTERS
0019.00 DATA RECORD IS RECORD1.
.....
*****END OF DATA*****

FMT SEQNBR   Cont Area-A Area-B
CB _____ - _____

```

Figure 8-1. Relationship between COBOL Coding Form and an SEU Display

- Any time a source line is entered or changed, up to 45 lines of source may be syntax checked as one unit. The length of a single unit of syntax checking is determined by extending from an entered or changed line as follows:
 - A unit of syntax checking extends towards the beginning of the source member until the first source line, or a line that ends in a period is encountered.
 - A unit of syntax checking extends towards the end of the source member until the last source line, or a line that ends in a period is encountered.

- If this unit spans more than 45 source lines (not including comment lines), the system responds with a message.
- If there is an error in a unit of syntax checking, the entire unit is presented in reverse image. The message at the bottom of the display screen refers to the first error in the unit.
- No compiler options are honored to control syntax checking.

For example, the syntax checker accepts either quotes or apostrophes as nonnumeric delimiters as long as they are not mixed within one unit of syntax checking. It does not check whether the delimiter is the one specified in the CRTCLPGM command or in the PROCESS statement.

- The first sentence following any of the paragraph headers listed below must begin on the same line as the paragraph header:

```
PROGRAM-ID.  
AUTHOR.  
INSTALLATION.  
DATE-WRITTEN.  
DATE-COMPILED.  
SECURITY.  
SOURCE-COMPUTER.  
OBJECT-COMPUTER.  
SPECIAL-NAMES.
```

- Character replacement specified by the CURRENCY and DECIMAL-POINT clauses of the SPECIAL-NAMES paragraph is not honored during interactive syntax checking.

For a complete description of how to enter a source program using SEU, refer to the *SEU Reference Manual and User's Guide*.

Using SEU to Browse through a Compiler Listing

SEU lets the user browse through a compiler listing that is on an output queue. The following shows SEU's split-edit display that lets the user browse through the listing from a work station.

```

EDIT   US W:1   Mbr: CBLEXMPL           Scan: _____
FMT CB .....-A+++B+++++Pgm-i
0012.00      INPUT OUTPUT SECTION.
0013.00      FILE-CONTROL.
0014.00      SELECT FILE1 ASSIGN TO PRINTER-QSYSVRT
0015.00      ORGANIZATION IS SEQUENTIAL.
0016.00      DATA DIVISION.
0017.00      FILE SECTION.
0018.00      FD FILE1
0019.00      RECORD CONTAINS 56 CHARACTERS
0020.00      LABEL RECORDS ARE OMITTED
0021.00      DATA RECORD IS REB-1.
0022.00      01 REC-1 PIC X(56).
-----
BROWSE Mbr:SPOOLFILE W:1   Pos: _____ Scan: *
30 003100 STOP RUN.
          * * * * * E N D O F S O U R C E * * * * *
5714CB1          COBOL MESSAGES          VERIFY
STMT SEQNBR MSGID SEV TEXT
* 21 002100 CBL1327 30 'REB-1' not previously defined. Clause ignored.
          MESSAGE SUMMARY
TOTAL   INFO(0-4)  WARNING(5-19)  ERROR(20-29)  SEVERE(30-39)  TERMIN
  1           0           0           0           1

```

Source Statements

Compiler Listing

An error is found.

An * requests a scan for compiler errors.

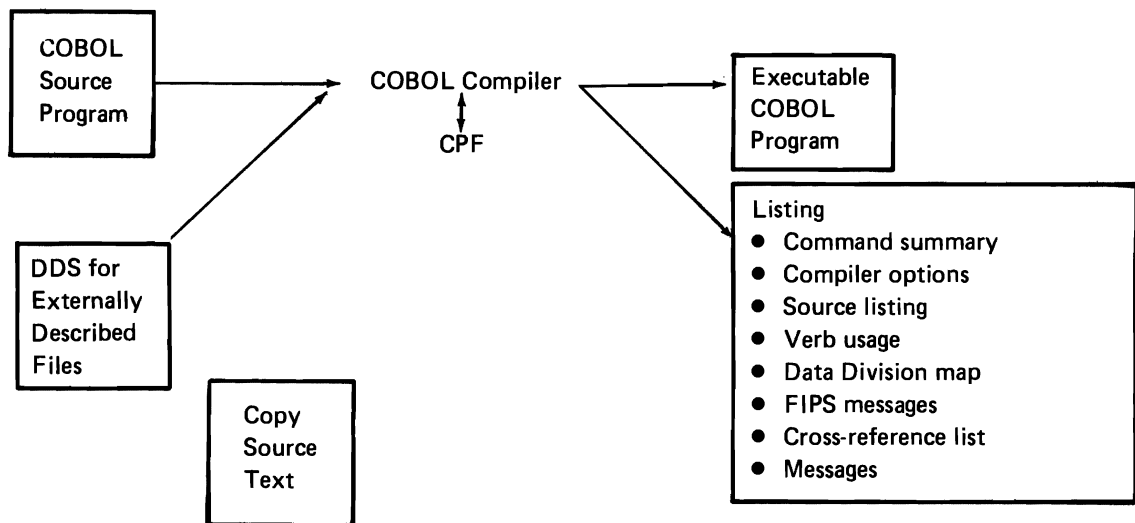
While browsing, the user can:

- Scan for errors
- Correct source statements that have errors.

For complete information on browsing through a compiler listing, see the *SEU Reference Manual and User's Guide*.

Compiling the Source Program

After you have entered the source program into the system, you need to compile the source program. This process of compilation is done by the COBOL compiler, which is part of the COBOL Licensed Program. To compile a COBOL program, you must use the CRTCLPGM (Create COBOL Program) command, and the libraries QCBL and QTEMP must be in the library list. If the COPY statement, DDS or DD format is used by the program, the Control Program Facility (CPF) provides information about the externally described files to the compiled program. The compiler is invoked to create an executable COBOL program and a listing:



The compiler syntax checks the COBOL source program line by line and the interrelationships between the lines.

Compiler Options

You can specify various compiler options either by using the CRTCLPGM command OPTION parameter or by using the PROCESS statement with the desired options. Any options specified in the PROCESS statement override the corresponding options on the CRTCLPGM command. The PROCESS statement is discussed later in this chapter.

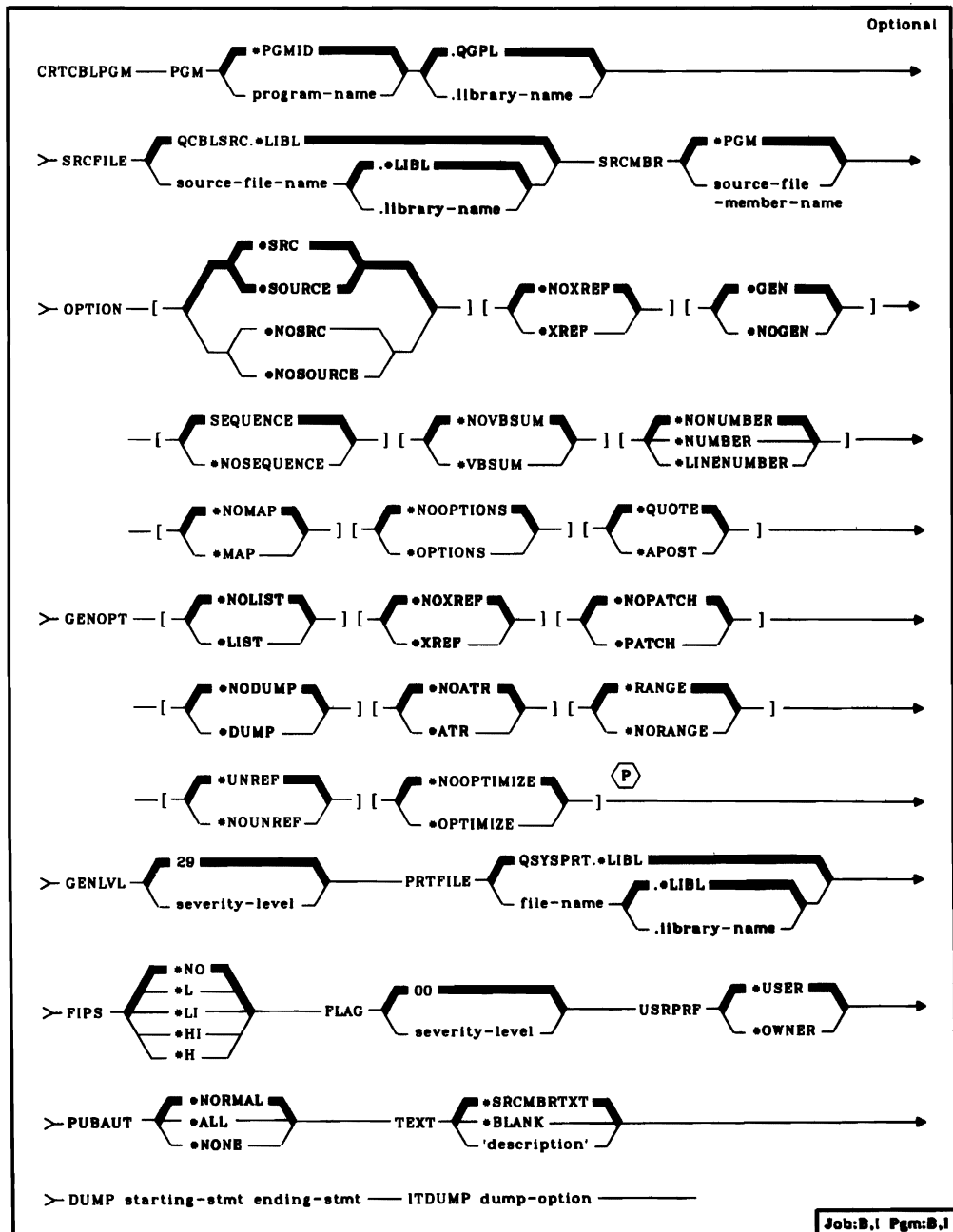
Create COBOL Program Command

To compile a COBOL source program into an executable program, you must enter the CRTCLPGM (Create COBOL Program) command that invokes the COBOL compiler. The command is valid in batch and interactive jobs, or from other programs.

If the COBOL compiler terminates, the escape message CBL9001 is issued. A control language program can monitor for this exception by using the control language MONMSG (Monitor Message) command.

All object names specified on the CRTCLPGM command must be composed of alphanumeric characters, the first of which must be alphabetic. The length of the names cannot exceed 10 characters. See the *CL Reference Manual* for a detailed description of CPF object naming rules and for a complete description of CPF command syntax.

The command syntax is as follows. The defaults are indicated by heavier branch lines. The brackets indicate that the options can be specified in any order.



When the CRTCLP command is issued in a CL program, concatenation expressions can be used for all parameter values. See the *CL Reference Manual* for more information about concatenation expressions.

Programming Note: The number of entries in the Object Definition Table (ODT) and the amount of storage required by a COBOL program varies with the number and kinds of COBOL statements used in the program. A combination of these factors can cause System/38 internal size limits for the program to be exceeded. If this occurs, the *NOUNREF option can be tried. If the problem persists, the program must be rewritten.

When the *NOUNREF option is specified, only names that are referenced or are needed for data structuring are defined. This option is useful when the program contains many unreferenced identifiers.

The description of the parameters follows (the defaults are explained first and are underscored):

PGM Parameter: Specifies the qualified name by which the compiled COBOL program can be known and the library in which the compiled program is to be located.

*PGMID: The name specified as the PROGRAM-ID is used.

program-name: Specifies the name by which the compiled COBOL program is known. If program-name is specified for the PGM parameter, the first program in the batch job has that name, and all other programs use the name specified in the PROGRAM-ID paragraph in the source program.

QGPL: Name of the library in which the created program is stored if no library-name is specified.

library-name: Specifies the name of the library in which the created program is stored.

SRCFILE Parameter: Specifies the name of the source file that contains the COBOL source to be compiled.

QCBLSRC: Specifies that the IBM-supplied source file, QCBLSRC, contains the COBOL source to be compiled.

qualified-source-file-name: Enter the qualified name of the source file that contains the COBOL source to be compiled. If no library qualifier is given, *LIBL is used to find the file. This source file should have a record length of 92.

SRCMBR Parameter: Specifies the name of the member of the source file that contains the COBOL source to be compiled. This parameter can be specified only if the source-file-name in the SRCFILE parameter is a data base file.

*PGM: The COBOL source to be compiled is in the member of the source file that has the same name as that specified for the compiled program in the PGM parameter. If *PGMID is specified for the PGM parameter, the SRCMBR parameter is not used. For a data base source file, the first member is used.

source-file-member-name: Enter the name of the member that contains the COBOL source.

OPTION Parameter: Specifies the options to use when the COBOL source is compiled.

*SOURCE or *SRC: The compiler produces a source listing, consisting of the COBOL source input and all compile-time error messages.

*NOSOURCE or *NOSRC: The compiler does not produce a source listing.

*NOXREF: The compiler does not produce a cross-reference listing for the source program.

*XREF: The compiler produces a cross-reference listing for the source program.

*GEN: The compiler creates an executable program after the program is compiled.

*NOGEN: The compiler does not create an executable program after the program is compiled.

*SEQUENCE: The reference numbers are checked for sequence errors. Sequence errors do not occur if the *LINENUMBER option is specified.

*NOSEQUENCE: The reference numbers are not checked for sequence errors.

*NOVBSUM: Verb usage counts are not printed.

*VBSUM: Verb usage counts are printed.

*NONUMBER: The source file sequence numbers are used for reference numbers.

*NUMBER: The user-supplied sequence numbers (columns 1 through 6) are used for reference numbers.

*LINENUMBER: The compiler generated sequence numbers are used for reference numbers. This option combines program source code and source code introduced by COPY statements into one consecutively numbered sequence. Use this option if you specify FIPS flagging.

*NOMAP: The compiler does not list the Data Division map.

*MAP: The compiler lists the Data Division map.

*NOOPTIONS: Options in effect are not listed for this compilation.

*OPTIONS: Options in effect are listed for this compilation.

*QUOTE: Specifies that the delimiter " is used for nonnumeric literals and Boolean literals. This also specifies that the value of the figurative constant QUOTE has the EBCDIC value of a quote.

*APOST: Specifies that the delimiter ' is used for nonnumeric literals and Boolean literals. This also specifies that the value of the figurative constant QUOTE has the EBCDIC value of an apostrophe.

GENOPT Parameter: Specifies the options to use when the executable program is created. The listings could be required if a problem occurs in COBOL.

*NOLIST: No IRP, associated hexadecimal code or error messages are listed.

*LIST: The IRP, its associated hexadecimal code, and any error messages are listed.

*NOXREF: A cross-reference listing of all objects defined in the IRP is not produced.

*XREF: A cross-reference listing of all objects defined in the IRP is produced.

*NOPATCH: Space is not reserved in the compiled program for a program patch area.

*PATCH: Space is reserved in the compiled program for a program patch area. The program patch area can be used for debugging purposes.

*NODUMP: The program template is not listed.

*DUMP: The program template is listed.

*NOATR: The attributes for the IRP source are not listed.

*ATR: The attributes for the IRP source are listed.

*RANGE: Execution time checks are performed for subscript ranges, but not index ranges. Checks are also performed for substring operations in compiler-generated code.

*NORANGE: Execution-time checks are not performed.

*UNREF: Unreferenced data items are included in the compiled program.

*NOUNREF: Unreferenced data items are not included in the compiled program. This reduces the number of ODT (object definition table) entries used, allowing a larger program to be compiled. The unreferenced data items still appear in the cross-reference listings produced by specifying OPTION (*REF).

*NOOPTIMIZE: The compiler performs only standard optimizations for the program.

*OPTIMIZE: The compiler generates a program for possibly more efficient execution that will possibly require less storage. However, specifying *OPTIMIZE can substantially increase the time required to compile a program.

GENLVL Parameter: Specifies when a program is generated. A severity-level value, corresponding to the severity level of the messages produced during compilation, can be specified in this parameter. If errors occur in a program with a severity level greater than the value specified in this parameter, an executable program is not generated. For example, if you do not want a program generated if you have messages with a severity level of 20 or greater, specify 19 in this parameter.

29: If a severity-level value is not specified, the default severity-level is 29.

severity-level: A two-digit number, 00 through 29, can be specified.

PRTFILE Parameter: Specifies the name of the file to which the compiler listing is directed and the library in which the file is located. The file should have a minimum record length of 132. If a file with a record length less than 132 is specified, information is lost.

QSYSPRT: If a file-name is not specified, the compiler listing is directed to the IBM-supplied file, QSYSPRT.

file-name: Enter the name of the file to which the compiler listing is directed.

*LIBL: If a library-name is not specified, the system searches the library list, *LIBL, to find the library in which the file is located.

.library-name: Enter the name of the library in which the file is located.

FIPS Parameter: The source program is FIPS flagged for the following specified level. (Select the *LINENUMBER option to ensure that the reference numbers used in the FIPS flagging messages are unique.)

*NO: The source program is not FIPS flagged.

*L: FIPS flag for low level and higher.

*LI: FIPS flag for low-intermediate level and higher.

*HI: FIPS flag for high-intermediate level and higher.

*H: FIPS flag for high level.

FLAG Parameter: Specifies the minimum severity level of messages to be printed.

00: All messages are to be printed.

severity-level: Enter a two-digit number that specifies the minimum severity level of messages that are to be printed. Messages that have severity levels of the specified value or higher are listed.

USRPRF Parameter: Specifies under which user profile the compiled COBOL program is to be executed. The profile of either the program owner or the program user is used to execute the program and control which objects can be used by the program (including what authority the program has for each object).

*USER: The program user's user profile is to be used when the program is executed.

*OWNER: The user profiles of both the program's owner and user are to be used when the program is executed. The collective sets of object

authority in both user profiles are to be used to find and access objects during the program's execution. Any objects that are created during the program are owned by the program's user.

Note: Specify the USRPRF parameter to reflect the security requirements of your installation. The security facilities available on System/38 are described in detail in the *CPF Programmer's Guide* and the *CL Reference Manual*.

PUBAUT Parameter: Specifies what authority for the program and its description is being granted to the public. The authority can be altered for all or for specified users after program creation through the GRTOBJAUT (Grant Object Authority) or RVKOBJAUT (Revoke Object Authority) commands. (For further information on these commands, see the *CL Reference Manual*.)

*NORMAL: The public has only operational rights for the compiled program. Any user can execute the program, but cannot change it or debug it.

*ALL: The public has complete authority for the program.

*NONE: The public cannot use the program.

Note: Specify the PUBAUT parameter to reflect the security requirements of your installation. The security facilities available on System/38 are described in detail in the *CPF Programmer's Guide* and the *CL Reference Manual*.

TEXT Parameter: Lets the user enter text that briefly describes the program and its function.

*SRCMBRTXT: Indicates that the text for the object being created is to be the same as the text for the data base file member containing the COBOL source. If the source comes from a device or in-line file, specifying *SRCMBRTXT has the same effect as specifying *BLANK.

*BLANK: No text is specified.

'text': The text that briefly describes the program and its function can be a maximum of 50 characters in length and must be enclosed in apostrophes. The apostrophes are not part of the 50-character string.

ITDUMP (n) Parameter: An IBM debugging aid. Causes the compiler to dump the internal text at various times during the compilation. Refer to Appendix J for values allowed for n. (For IBM service personnel.)

DUMP Parameter: An IBM COBOL debugging aid. (For IBM service personnel.)

PROCESS Statement

The PROCESS statement also lets you specify compile-time options. Options specified in the PROCESS statement override corresponding options specified in the CRTCLPGM command. The PROCESS statement must be placed before the first source statement in the COBOL program immediately preceding the IDENTIFICATION DIVISION header.

The format of the PROCESS statement is as follows:

Format

```
PROCESS option-1 [option-2] . . . [option-n] [.]
```

The following rules apply to the PROCESS statement:

- The word PROCESS and all options must appear within positions 8 through 72. Position 7 must be left blank. The remaining positions can be used as in COBOL source statements, positions 1 through 6 for sequence numbers, positions 73 through 80 for identification purposes.
- Options must be separated by one or more blanks and/or commas.
- Options can appear in any order. If conflicting options are specified, for example, LIST and NOLIST, the last option encountered takes precedence.
- If the option keyword is correct and the suboption(s) is in error, the default suboption(s) is assumed.
- The PROCESS statement begins with the word PROCESS. Options can appear on more than one line; however, only the first line can contain the word PROCESS.

The allowable options for the PROCESS statement are described as follows. Defaults are underlined.

SOURCE or SRC: The compiler produces a source listing, consisting of the COBOL source input and all compile-time error messages.

NOSOURCE or NOSRC: The compiler does not produce a source listing.

NOXREF: The compiler does not produce a cross-reference listing for the source program.

XREF: The compiler produces a cross-reference listing for the source program.

GEN: The compiler creates an executable program after the program is compiled.

NOGEN: The compiler does not create an executable program after the program is compiled.

SEQUENCE: The reference numbers are checked for sequence errors. Sequence errors do not occur if the **LINENUMBER** option is specified.

NOSEQUENCE: The reference numbers are not checked for sequence errors. Since **SEQUENCE** is the default option, sequence errors are flagged until the **NOSEQUENCE** option is recognized. When **NOSEQUENCE** is the last item specified on a record, sequence checking is in effect between that record and the next record.

NOVBSUM: Verb usage counts are not printed.

VBSUM: Verb usage counts are printed.

NONUMBER: The source file sequence numbers are used for reference numbers.

NUMBER: The user-supplied sequence numbers (columns 1 through 6) are used for reference numbers.

LINENUMBER: The compiler generated sequence numbers are used for reference numbers. The option combines program source code and source code introduced by **COPY** statements into one consecutively numbered sequence. Use this option if you specify **FIPS** flagging.

NOMAP: The compiler does not list the Data Division map.

MAP: The compiler lists the Data Division map.

NOOPTIONS: Options in effect are not listed for this compilation.

OPTIONS: Options in effect are listed for this compilation.

QUOTE: Specifies the delimiter " is used for nonnumeric literals and Boolean literals. This also specifies that the value of the figurative constant **QUOTE** has the EBCDIC value of a quote.

APOST: Specifies that the delimiter ' is used for nonnumeric literals and Boolean literals. This also specifies that the value of the figurative constant **QUOTE** has the EBCDIC value of an apostrophe.

NOLIST: No IRP, associated hexadecimal code, or error messages are listed.

LIST: The IRP, its associated hexadecimal code, and any error messages are listed.

GENLVL (n): Specifies when a program is generated. A severity-level value, corresponding to the severity level of the messages produced during compilation, can be specified in this parameter. If errors occur in a program with a severity level greater than the value specified in this parameter, the compilation is terminated. For example, if you do not want a program generated if you have messages with a severity level of 30 or greater, specify 29 in this parameter. The default value is 29.

FIPS: The source program is FIPS flagged for the following specified level. (Select the LINENUMBER option to ensure that the reference numbers used in the FIPS flagging messages are unique.)

NO: The source program is not FIPS flagged.

L: FIPS flag for low level and higher.

LI: FIPS flag for low-intermediate level and higher.

HI: FIPS flag for high-intermediate level and higher.

H: FIPS flag for high level.

FLAG: Specifies the minimum severity level of messages to be printed.

00: All messages are to be printed.

severity-level: Enter a two-digit number that specifies the minimum severity level of messages that are to be printed. Messages that have severity levels of the specified value or higher are listed.

Batch Compiles

The PROCESS statement is used to separate multiple programs and/or subprograms to be compiled with a single invocation of the compiler. In the batch compile environment, all compiler options specified on the CRTCLPGM command statement, plus all default options, plus the options specified on the last PROCESS statement will be in effect for the compilation. All compiler output is directed to the destinations as specified by the CRTCLPGM command statement.

All executable programs are stored in the library specified on the PGM parameter. If program-name is specified for the PGM parameter, the first program in the batch job has that name, and all other programs use the name specified in the PROGRAM-ID paragraph in the source program.

Using COPY within the PROCESS Statement

The COBOL COPY statement can be used within the PROCESS statement to retrieve compiler options previously stored in a source library and include them in the PROCESS statement. COPY can be used to include options that override those specified as defaults by the compiler. Any PROCESS statement options can be retrieved with the COPY statement.

Compiler options can both precede and follow the COPY statement within the PROCESS statement. The last encountered occurrence of an option overrides all preceding occurrences of that option.

The following example shows the use of the COPY statement within the PROCESS statement. The COPY statement must be followed by a period. Notice also that in this example, NOMAP overrides the corresponding option in the library member:

```
000001 PROCESS XREF           MYPROG
000002 COPY DEFLTS           MYPROG
                                MAP, SOURCE, LIST     DEFLTS
000004 NOMAP, FLAG20        MYPROG
000010 IDENTIFICATION DIVISION. MYPROG
```

Compiler Output

The output of the compilation job step can include:

- A summary of command options.
- An options listing: A listing of options in effect for the compilation.
- A source listing: A listing of the statements contained in the source program.
- A verb usage listing: A listing of the COBOL verbs and the number of times each verb is used.
- A Data Division map: A glossary of compiler-generated information about the data. Also included is a mapping of user-supplied names to compiler-generated internal names.
- FIPS messages: A list of all FIPS messages for the requested FIPS level and above.
- A cross-reference list.
- Compiler messages (including diagnostic statistics).
- Compilation statistics.
- A listing of the generated program in symbolic form.
- An executable program.

The presence or absence of some of these types of compiler output is determined by options specified on the PROCESS statement or through the CRTCLPGM command. The level of diagnostic messages printed depends upon the FLAG option. Page ejection of the source program listing is obtained by placing the slash / character in the continuation area of a comment line.

Figures 8-2 through 8-8 illustrate the compiler output produced for the sample program. References to the figures are made throughout the following text. The letters in text correspond to the letters in the figures and reference an area of the output as it is being discussed.

Command Summary

This summary, which is output after compilation, lists all options specified in the CRTCLPGM command statement and as modified by the PROCESS statement.

Command Summary

This summary, which is output after compilation, lists all options specified in the CRTCLPGM command statement and as modified by the PROCESS statement.

Compiler Options

The PROCESS statement, if specified, is printed immediately. Figure 8-2 is a list of all options in effect for the compilation of the sample program. Compiler options are listed at the beginning of all compiler output when the OPTIONS option is specified.

```
                                COBOL SOURCE LISTING
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME

1 000100 PROCESS OPTIONS,LIST,MAP,XREF,FLAG(00),FIPS(L),VBSUM
      COBOL COMPILER OPTIONS IN EFFECT

      OPTIONS
      SOURCE
      XREF
      MAP
      VBSUM
      NONUMBER
      SEQUENCE
      GEN
      GENLVL(29)
      FLAG( 0)
      FIPS(L)
      QUOTE

      COBOL GENERATION OPTIONS IN EFFECT

      LIST
      UNREF
      RANGE
      NOATR
      NOXREF
      NODUMP
      NOPATCH
```

Figure 8-2. Compiler Options

Source Listing

Figure 8-3 illustrates a source listing. The statements in the source program are listed exactly as submitted. The source is not listed if the NOSOURCE option is specified.

All compiler output pages after the page where the PROGRAM-ID paragraph is listed have the program-id name listed in the heading prior to the date field. Figure 8-3 displays the following fields:

- A** *Compiler-generated statement number:* The numbers appear to the left of the source program listing. These numbers are referenced in all compiler output listings except for FIPS messages listings. A statement number can span several lines, and a line can contain more than one statement.

- B** *Reference number:* The numbers appear to the left of the source statements. The numbers that appear in this field and the column heading (shown as SEQNBR in this listing) are determined by an option specified in the CRTCLPGM command or in the PROCESS statement, as shown in the following table.

Option	Heading	Origin
NONUMBER	SEQNBR	Source file sequence numbers.
NUMBER	NUMBER	Standard COBOL sequence numbers.
LINENUMBER	LINNBR	Compiler generated sequence numbers.

- C** *Sequence error indicator column:* An S in this column indicates that the line is out of sequence. Sequence checking is performed on the reference number field only if the SEQUENCE option is specified.
- D** *Copyname:* The copyname, as specified in the COBOL COPY statement, is listed here for all records included in the source program by that COPY statement. If the DDS-ALL-FORMATS phrase is used, the name '<--ALL-FMTS' appears under copyname, and each individual format is terminated with the compiler-generated comment line 'END OF: format-name -->'.
OF: format-name -->'
- E** *Change/date field:* The date the line was last modified is listed here.

COBOL SOURCE LISTING

STMT	SEQNBR	-A	B	...	2	...	3	...	4	...	5	...	6	...	7	IDENTFCN	S	COPYNAME	CHG/DATE	
A	B																			
2	000200																			
3	000300																			
4	000400																			
5	000500																			
6	000600																			
7	000700																			
8	000800																			
9	000900																			
10	001000																			
11	001100																			
12	001200																			
13	001300																			
14	001400																			
15	001500																			
16	001600																			
17	001700																			
18	001800																			
19	001900																			
20	002000																			
21	002100																			
22	002200																			
23	002300																			
24	002400																			
25	002500																			
26	002600																			
27	002700																			
28	002800																			
29	002900																			
30	003000																			
31	003100																			
32	003200																			
33	003300																			
34	003400																			
35	003500																			
36	003600																			
37	003700																			
38	003800																			
40	+000100																			
41	+000200																			
42	+000300																			
43	+000400																			
44	+000500																			
45	+000600																			
46	+000700																			
47	+000800																			
48	+000900																			
49	003900																			
004000	BEGIN.																			
004100*	*****																			
004200**	THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO BE CREATED**																			
004300**	AND INITIALIZES COUNTERS. ***																			
004400*	*****																			
50	004500																			
51	004600																			
004700*	*****																			
004800**	THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE ***																			
004900**	CONTAINED IN THE FILE, WRITES THEM ON DISK, AND DISPLAYS ***																			
005000**	THEM. ***																			
005100*	*****																			
52	005200																			
54	005300																			
55	005400																			
56	005500																			
57	005600																			
58	005700																			
59	005800																			
005900*	*****																			
006000**	THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS ***																			
006100**	INPUT. ***																			
006200*	*****																			
006300	STEP-5.																			
60	006400																			
61	006500																			
006600*	*****																			
006700**	THE FOLLOWING READS BACK THE FILE AND SINGLES OUT EMPLOYEES**																			
006800**	WITH NO DEPENDENTS. ***																			
006900*	*****																			
007000	STEP-6.																			
62	007100																			
63	007200																			
007300	STEP-7.																			
64	007400																			
65	007500																			
66	007600																			
007700	STEP-8.																			
67	007800																			
68	007900																			

WRKRCD
WRKRCD
WRKRCD
WRKRCD
WRKRCD
WRKRCD
WRKRCD
WRKRCD

Figure 8-3. Source Listing

Verb Usage by Count Listing

Figure 8-4 shows the alphabetic list that is produced of all verbs used in the source program. A count of how many times each verb was used is also included. This listing is produced when the VBSUM option is specified.

VERB	COUNT
ADD	2
CLOSE	2
DISPLAY	1
GO	2
IF	1
MOVE	5
OPEN	2
PERFORM	1
READ	1
STOP	1
WRITE	1

Figure 8-4. Verb Usage by Count Listing

Data Division Map

The Data Division map, Figure 8-5, is listed when the MAP option is specified. The Data Division map contains information about names in the COBOL source program. The number of bytes required for the File Section and Working-Storage Section is given at the end of the Data Division map. Figure 8-5 displays the following fields:

- F** *Statement number:* The compiler-generated statement number where the data item was defined is listed for each data item in the Data Division map.
- G** *Level of data item:* The level-number of the data item, as specified in the source program, is listed here. Index-names are identified by an *IX* in the level-number and a blank type field.
- H** *Source name:* The data-name as specified in the source program is listed here.
- I** *Section:* The section in which the item was defined is shown here through the use of the following codes:
 - FS File Section
 - WS Working-Storage Section
 - LS Linkage Section
 - SM Sort/Merge Section
 - SR Special Register
- J** *Displacement:* The offset, in bytes, of the item from the level-01 group item is given here.
- K** *Length:* The decimal length of the item is listed here.

- L** *Type:* The data class type for the item is shown here through the use of the following codes:

GROUP	Group Item
A	Alphabetic
AE	Alphabetic edited
AN	Alphanumeric
ANE	Alphanumeric edited
INDEX	Index data item (USAGE INDEX)
BOOLN	Boolean
ZONED	Zoned decimal
PACKED	Packed decimal (COMP or COMP-3)
BINARY	Binary (COMP-4)
NE	Numeric edited

- M** *Internal name:* The compiler-generated internal names are listed here and are assigned as follows:

File-names The internal name uses the form .Fnn, where .F indicates a file-name, and nn is a unique two-digit number.

Data-names The internal name uses the form .Dxxxxxx, where .D indicates a data-name or index-name, and xxxxxx is a unique six-digit hex value. These names appear on the PRM listing if generated.

- N** *Attributes:* The attributes of the item are listed here as follows:

- For files, the following information can be given:
 - Device type
 - ORGANIZATION
 - ACCESS MODE
 - BLOCK CONTAINS information
 - RECORD CONTAINS information
 - LABEL information
 - RERUN specified is indicated
 - SAME AREA specified is indicated
 - CODE-SET specified is indicated
 - SAME RECORD AREA specified is indicated
 - LINAGE specified is indicated.
- For data items, the attributes indicate whether the following information was specified for the item:
 - REDEFINES
 - VALUE
 - JUSTIFIED
 - SYNCHRONIZED
 - BLANK WHEN ZERO
 - SIGN IS LEADING
 - SIGN IS LEADING SEPARATE
 - SIGN IS SEPARATE
 - INDICATORS.

- For table items, the dimensions for the item are listed here in the form dim (). For each dimension, a maximum OCCURS value is given. When a dimension is variable, it is listed as such, giving the lowest and highest OCCURS values.

STMT	LVL	SOURCE NAME	SECTION	DISP	LEN	TYPE	E-NAME	ATTRIBUTES
F	G	H	I	J	K	L	M	N
18	FD	FILE-1	FS				.F01	DEVICE DISK, ORGANIZATION SEQUENTIAL, ACCESS SEQUENTIAL, BLOCK CONTAINS 20 CHARACTERS, RECORD CONTAINS 20 CHARACTERS, LABEL RECORDS STANDARD
22	01	RECORD-1	FS	000000	20	GROUP	.D005394	
23	02	FIELD-A	FS	000000	20	AN	.D0053E8	
24	FD	FILE-2	FS				.F02	DEVICE DISK, ORGANIZATION SEQUENTIAL, ACCESS SEQUENTIAL, BLOCK CONTAINS 20 CHARACTERS, RECORD CONTAINS 20 CHARACTERS, LABEL RECORDS STANDARD
28	01	RECORD-2	FS	000000	20	GROUP	.D0054E4	
29	02	FIELD-A	FS	000000	20	AN	.D005538	
31	01	FILLER	WS	000000	56	GROUP	.D005582	
32	02	KOUNT	WS	000000	2	PACKED	.D0055D4	
33	02	ALPHABET	WS	000000	26	AN	.D005636	VALUE
34	02	ALPHA	WS	000002	1	AN	.D005692	REDEFINES .D005636, DIMENSION(26)
35	02	NUMBR	WS	000028	2	PACKED	.D005712	
36	02	DEPENDENTS	WS	000000	26	AN	.D005774	VALUE
37	02	DEPEND	WS	000030	1	AN	.D0057EE	REDEFINES .D005774, DIMENSION(26)
40	01	WORK-RECORD	WS	000000	21	GROUP	.D00584E	
41	02	NAME-FIELD	WS	000000	1	AN	.D0058A6	
42	02	FILLER	WS	000001	1	AN	.D0058FC	VALUE
43	02	RECORD-NO	WS	000002	5	ZONED	.D005956	
44	02	FILLER	WS	000007	1	AN	.D00599E	VALUE
45	02	LOCATION	WS	000008	3	A	.D005A18	VALUE
46	02	FILLER	WS	000011	1	AN	.D005A7E	VALUE
47	02	NO-OF-DEPENDENTS	WS	000012	2	AN	.D005AD8	
48	02	FILLER	WS	000014	7	AN	.D005B34	VALUE

FILE SECTION USES 40 BYTES OF STORAGE
 WORKING-STORAGE SECTION USES 77 BYTES OF STORAGE

Figure 8-5. Data Division Map

FIPS Messages

The FIPS messages, Figure 8-6, are listed when the FIPS option is specified. Only messages for the requested FIPS level and above are listed. Figure 8-6 displays the following fields:

FIPS-ID	DESCRIPTION AND SEQUENCE NUMBERS FLAGGED
Q	P
CBL8200	FOLLOWING ITEMS ONLY VALID AT FIPS LEVEL 2 OR HIGHER.
CBL8201	COPY STATEMENT. 38
CBL8300	FOLLOWING ITEMS VALID ONLY AT FIPS LEVEL 3 OR HIGHER.
CBL8302	PLURAL FORM OF FIGURATIVE CONSTANTS USED. 48
CBL8303	DATE-COMPILED PARAGRAPH. 7
CBL8350	UNTIL PHRASE OF PERFORM STATEMENT. 59
CBL8500	FOLLOWING ITEMS ARE IMPLEMENTOR-DEFINED OR ARE IHM EXTENSIONS.
CBL8504	ASSIGN TO SYSTEM-NAME-1 CLAUSE OR SYSTEM-NAME CLAUSE. 14 15
CBL8506	FILLER USED AS GROUP ITEM. 31
CBL8518	USAGE IS COMPUTATIONAL-3. 32 35
CBL8561	COPY STATEMENT WITH DEFAULT LIBRARY ASSUMED. 38
R	
10 FIPS VIOLATIONS FLAGGED.	

Figure 8-6. FIPS Messages

- Q** FIPS-ID: This field lists the FIPS message number.
- P** *Description and reference numbers flagged:* This field lists a description of the condition flagged, followed by a list of the reference numbers from the source program where this condition is found. The type of reference numbers used, and their names in the heading (shown as SEQUENCE NUMBERS in this listing) are determined by an option specified in the CRTCBPLPGM command or in the PROCESS statement, as shown in the following table.

Option	Heading
NONUMBER	DESCRIPTION AND SEQUENCE NUMBERS FLAGGED
NUMBER	DESCRIPTION AND USER-SUPPLIED NUMBERS FLAGGED
LINENUMBER	DESCRIPTION AND LINENUMBERS FLAGGED

- Q** *Items grouped by level:* These headings subdivide the FIPS messages by level.
- R** *FIPS violations flagged:* The total number of FIPS violations flagged is included at the end of the FIPS listing.

Cross-Reference List

The cross-reference list, Figure 8-7, is produced when the XREF option is specified. It provides a list of all data references and procedure-name references, by statement number, within the source program. Figure 8-7 displays the following fields:

COBOL CROSS REFERENCE LISTING		TESTPR			
NAMES (* = PROCEDURE-NAME)	DEFINED	REFERENCES (* = CHANGED)			
S	T	U			
ALPHA	34	54			
ALPHABET	33	34			
*BEGIN	50				
DEPEND	37	55			
DEPENDENTS	36	37			
*DUMMY-SECTION	50				
FIELD-A	23				
FIELD-A	29				
FILE-1	18	14	50	60	
FILE-2	24	15	61	62	67
KOUNT	32	51*	52*	54	55 59
LOCATION	45				
NAME-FIELD	41	54*			
NO-OF-DEPENDENTS	47	55*	64	65*	
NUMBR	35	51*	53*	56	
RECORD-NO	43	56*			
RECORD-1	22	21	58*		
RECORD-2	28	27			
*STEP-1	50				
*STEP-2	52	59			
*STEP-3	57	59			
*STEP-4	59				
*STEP-5	60				
*STEP-6	62	66			
*STEP-7	64				
*STEP-8	67	63			
WORK-RECORD	40	57	58	62*	

Figure 8-7. Cross-Reference List

- S** *Names field:* The data-name or procedure-name referenced is listed here. All procedure-names are flagged with an * before the name. The names are listed alphabetically.
- T** *Defined field:* The statement number where the name was defined within the source program is listed here.
- U** *References field:* All statement numbers are listed in the same sequence as the name is referenced in the source program. An * following a statement number indicates that the item was modified in that statement.

Messages

Figure 8-8 shows the messages that are generated during program compilation. The fields displayed are:

STMT	SEQNBR	MSGID	SEV	TEXT
(V)	(W)	(X)	(Y)	
* 43	+000400	CBL124 2	30	*S* USED INCORRECTLY IN PICTURE. ASSUMED S9(5).
* 50	004500	CBL0335 00		EMPTY PARAGRAPH OR SECTION PRECEDES *STEP-1* PARAGRAPH OR SECTION.
* 54	005300	CBL0010 10		*MOVE* INVALID IN AREA A. ACCEPTED AS IF IN AREA B.

MESSAGE SUMMARY

(Z) TOTAL	INFO(0-4)	WARNING(5-19)	ERROR(20-29)	SEVERE(30-39)	TERMINAL(40-99)
3	1	1	0	1	0

79 SOURCE RECORDS READ
 9 COPY RECORDS READ
 1 COPY MEMBERS PROCESSED
 0 SEQUENCE ERRORS
 30 WAS THE HIGHEST SEVERITY MESSAGE ISSUED

Figure 8-8. Diagnostic Messages

- (V) *Statement number:* This field lists the compiler-generated statement number associated with the statement in the source program for which the message was issued.¹
- (W) *Reference number:* The reference number is issued here.¹ The numbers that appear in this field and the column heading (shown here as SEQNBR) are determined by an option specified in the CRTCBPLPGM command or in the PROCESS statement, as shown in the following table.

Option	Heading	Origin
NONUMBER	SEQNBR	Source file sequence numbers
NUMBER	NUMBER	Standard COBOL sequence numbers
LINENUMBER	LINNBR	Compiler generated sequence numbers

When a message is issued for a record from a copy file, the number is preceded by a + or a -.

- (X) *MSGID and Severity Level:* These fields contain the message number and its associated severity level. Severity levels are defined as follows:

- 00 Informational
- 10 Warning
- 20 Conditional
- 30 Severe
- 40 Unrecoverable

- (Y) *Message:* The message identifies the condition and indicates the action taken by the compiler.

¹ The statement number and the reference number do not appear on certain messages that reference missing items. For example, if the PROGRAM-ID paragraph is missing, message CBL0031 appears on the listing with no statement or reference number listed.



Message statistics: This field lists the total number of messages and the number of messages by severity level.

The totals listed are the number of messages generated for each severity by the compiler and will not always be the number listed. For example, if `FLAG(10)` is specified, no messages of severity less than 10 are listed. However, the counts will indicate the number that would have been printed if they had not been suppressed.

How to Execute a COBOL Program

There are many ways to execute a COBOL program depending on how the program is written and who is using the program. See the *CPF Programmer's Guide* for the various ways to execute a COBOL program. The three most common ways are through:

- A control language `CALL` statement or the COBOL `CALL` statement
- An application-oriented menu
- A user-created command.

A control language `CALL` statement can be part of a batch job, entered interactively by a work station user, or included in a CL program. An example of a control language `CALL` statement is `CALL PAYROLL`. Payroll is the name of a COBOL program that is called and then executed. A COBOL program can call another program with the COBOL `CALL` statement (see Chapter 6). Another way to execute a COBOL program is through an application-oriented menu. The work station user can request an application-oriented menu and then select an option that will call the appropriate program. The following is an example of an application-oriented menu:

```
PAYROLL DEPARTMENT MENU

1. Inquire into employee master
2. Change employee master
3. Add new employee
4. Return

Option:__
```

This menu is normally displayed by a control language program in which each option calls a separate COBOL program. When a COBOL program is ended, the system returns control to whoever called the program. This could be a work station user, a CL program (such as the menu handling program), or another COBOL program.

You can also create a command yourself to execute a COBOL program by using a command definition. Refer to the *CPF Programmer's Guide* for information on using the command definition. For example, you can create a command, PAY, which calls a program, PAYROLL. A user-created command can be entered into a batch job, or it can be entered interactively by a work station user.

If a COBOL program terminates abnormally during execution, the escape message CBE9001 is issued. A control language program can monitor for this exception by using the MONMSG (Monitor Message) command.

If a program terminates other than by a STOP statement or falling through to the end of the program, the return code is set to 2. See the RTVJOBA and DSPJOB commands in the *CL Reference Manual* for more information about return codes.

When the user is executing a batch job that uses the ACCEPT statement, the input data is taken from the job stream. The data must be placed immediately following the CL CALL for the COBOL program. It is the user's responsibility not to request (through ACCEPTs) more data than is available. If more data is requested than is available, the CL statement following the data is treated as input data.

Programming Note: If more data is requested than is available, unpredictable results can occur.

When the user is executing a batch job that uses the ACCEPT statement, the input data is taken from the job stream. The data must be placed immediately following the CL CALL for the COBOL program. It is the user's responsibility to request (through ACCEPTs) the same amount of data as is available.

Programming Note: If more data is requested than is available, the CL statement following the data is treated as input data.

If more data is available than is requested, each extra line of data is treated as a CL command.

In either of the above cases, undesirable results can occur.



Chapter 9. Programmer's Guide Information

This chapter describes:

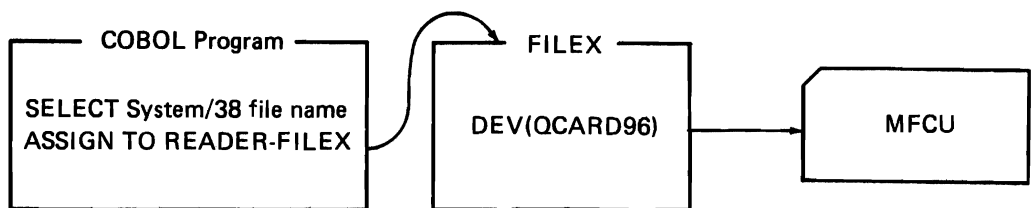
- The device-independent and device-dependent characteristics of COBOL on System/38.
- System/38 spooling functions.
- Level checking functions.
- File and record locking considerations.
- Unblocking and blocking records to improve performance.
- Multiple member processing.
- System override considerations.
- General information about the use of externally described files and program described files in the COBOL program.
- COBOL functions that relate specifically to a COBOL PRINTER device, and an MFCU device. Special considerations for DISK devices are also discussed.
- Commitment control considerations.
- General programming considerations.
- Recovery after a failure.
- Inter-Program Communications considerations.
- General information about the local data area available to a COBOL program.
- File considerations.

Device Independence/Device Dependence

The key element for all I-O operations on System/38 is the file. All files used on the system are defined to CPF. CPF maintains a description of each file that is accessed by a program when the file is used.

The files are kept online and serve as the connecting link between a program and the device used for I-O. The actual device association is made when the file is processed. In some instances, this type of I-O control allows the user to change the attribute of the file (and, in some cases, change the device) used in a program without changing the program.

In System/38 COBOL, the System/38 file name specified in the ASSIGN clause of the file control entry, rather than the device name specified, is used to point to the file. The System/38 file name points to the CPF file description:

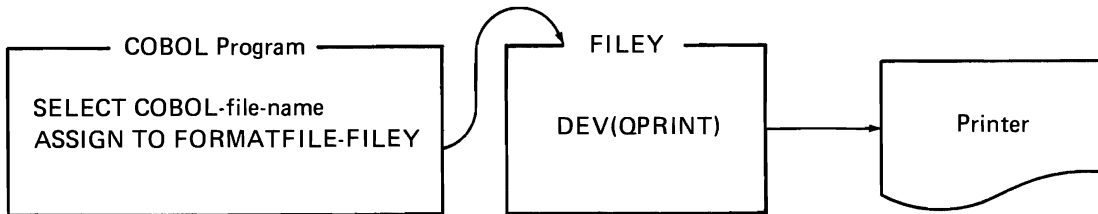


The COBOL device name in the ASSIGN clause defines the COBOL functions that can be performed on the selected file. At compilation time, certain COBOL functions are valid only for a specific COBOL device name; therefore, in this respect, COBOL is device dependent. The following are examples of device dependency:

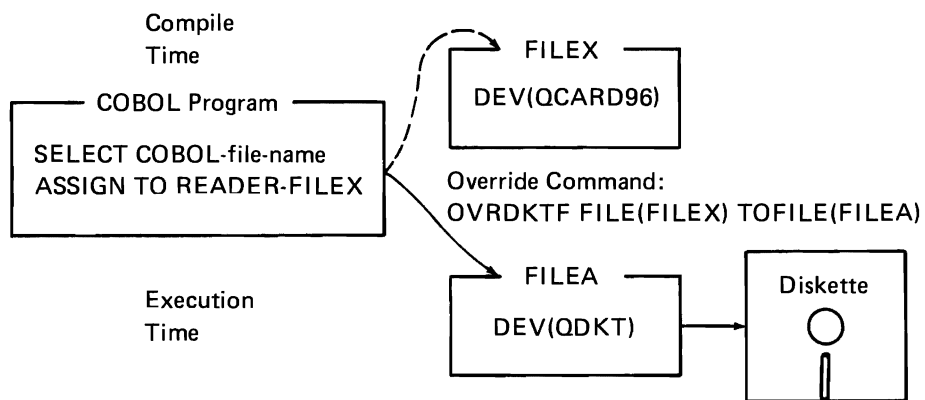
- SUBFILE operations are valid only for a WORKSTATION device.
- Indicators are valid only for WORKSTATION or FORMATFILE devices.
- LINAGE is valid only for the PRINTER device.
- Stacker select is valid only for the MFCU.
- OPEN INPUT WITH NO REWIND is valid only for a TAPEFILE device.

In the preceding example, stacker select specifications are valid in the program because the device name is READER.

For another example, assume that the file name FILEY is specified in the COBOL program with the FORMATFILE device. The device FORMATFILE is an independent device type; therefore, no line or page control specifications are valid in the COBOL program in the WRITE ADVANCING statement. When the program is executed, the actual I-O device is specified in the description of FILEY; for example, the device might be a printer, in which case only the default line and page control or those defined in the DDS would be used:



CPF commands can be used to override a parameter in the specified file description (such as specifying hopper 2 for a card input file instead of the default of hopper 1) or to redirect a file at compilation time or execution time. File redirection allows the user to specify one file at compilation time and another file at execution time:



In the preceding example, the Override to Diskette File command (OVRDKTF) allows the program to execute with an entirely different device file than was specified at compilation time.

Not all file redirections or overrides are valid. At execution time, checking occurs to ensure that the specifications within the COBOL program are valid for the file being processed. CPF allows some file redirections even if device specifics are contained in the program. For example, if the COBOL device name is PRINTER and the actual file the program uses is not a printer, CPF ignores the COBOL print spacing and skipping specifications.

There are other file redirections that CPF does not allow and that cause program termination. For example, if the COBOL device name is DATABASE or DISK and a keyed READ operation is specified in the program, the program is terminated if the actual file the program uses is not a disk or data base file. In addition, associated card files cannot be redirected because the compiler uses a single operation (PUTGET) to punch one record and read the next. This operation is valid only to a card device capable of both reading and punching.

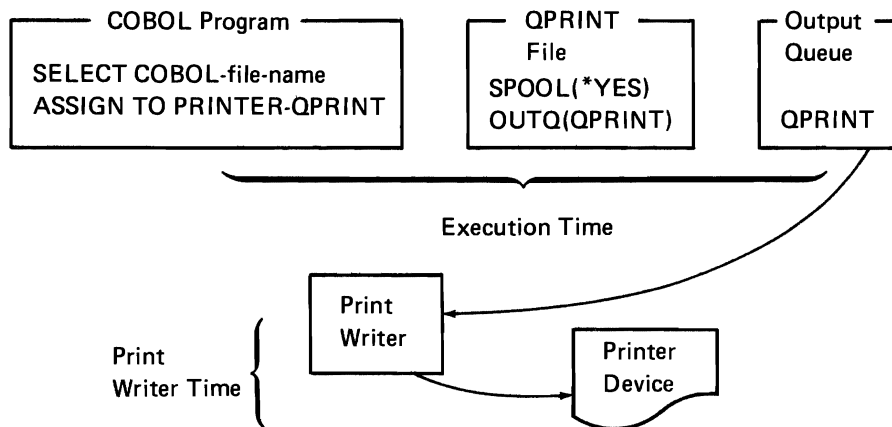
See the *CPF Programmer's Guide* for more detailed information on valid file redirections and file overrides.

Spooling

System/38 provides for the use of input and output spooling functions. Each System/38 file description contains a spool attribute that determines whether spooling is used for the file at execution time. The COBOL program is not aware that spooling is being used. The actual physical device from which a file is read or to which a file is written is determined by the spool reader or the spool writer. Associated card files in COBOL must not be spooled, or results are unpredictable. See the *CPF Programmer's Guide* for more detailed information on spooling.

Output Spool

Output spooling is valid for batch and interactive jobs. The description of the file that is specified in COBOL by the system-name contains the specification for spooling as shown in the following example:

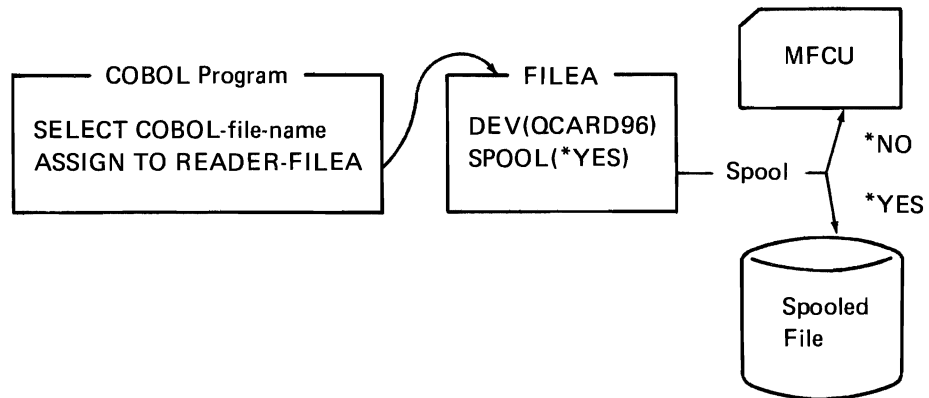


File override commands can be used at execution time to override the spooling options that are specified in the file description, such as the number of copies to be printed. In addition, System/38 spooling support allows a user to redirect a file after the program has executed. For example, the user can direct the printer output to a different device, such as a diskette.

Input Spool

Input spooling is valid only for inline data files in batch jobs. If the input data read by COBOL comes from a spooled file, COBOL is not aware of which device the data was spooled in from.

The data is read from a spooled inline file:



See the *CPF Programmer's Guide* for more information on inline data files.

Level Checking

When a COBOL program uses an externally described file, System/38 provides a level check function. This function ensures that the format has not changed since compilation time.

COBOL always provides the information required by level checking when an externally described file is used (when a record description was defined for the file by using the COPY statement, DDS or DD format). Only formats that are copied by the COPY statement, DDS or DD format, under the FD for a file are level checked. The level check function can be requested on the create, change, and override file commands. The default on the create file command is to request level checking. If level checking was requested, level checking occurs on a record format basis when the file is opened. If a level check error occurs, COBOL sets a file status of 90 at OPEN time.

If an existing format is used in a new file, any existing COBOL programs that use that format can still be used (assuming that no other conflicts such as a change of keys exist) without recompilation.

COBOL does not provide level checking for program described files.

For more information on how to specify level checking, see the *CL Reference Manual*.

File and Record Locking by COBOL

CPF allows a lock state (exclusive, exclusive allow read, shared for update, shared no update, or shared for read) to be placed on a file used during a routing step. The file can be allocated in such a manner with the Allocate Object (ALCOBJ) command. (For more information on allocating resources and the lock states, see the *CPF Programmer's Guide*.)

If an ALCOBJ command is not used for a routing step, COBOL places the following lock states on data base files when it opens the files:

OPEN Type	Lock State
INPUT	Shared-for-read
I-O	Shared-for-update
EXTEND	Shared-for-update
OUTPUT	Shared-for-update

The shared-for-read lock state allows another user to open the file with a lock state of shared-for-read, shared-for-update, shared-no-update, or exclusive-allow-read, but the user cannot specify the exclusive use of the file. The shared-for-update lock state allows another user to open the file with shared-for-read or shared-for-update lock state.

In order for programs to share a data base file, the file must be opened by the first program (the program with the highest invocation in the stack) for the maximum amount of functions to be performed by any subsequent programs that share the same file. If a subsequent program requests a function that was not specified by the first program, an abnormal termination occurs.

COBOL places an exclusive-allow-read lock state on device files. Another user can open the file with a shared-for-read lock state.

The lock state placed on the file by COBOL can be changed if you use the ALCOBJ command.

Releasing a Record Read for Update

When a data base record is read by COBOL and the file is opened for I-O, a lock is placed on that record so that it is not available to another program. COBOL releases the record from its locked state when the next successful I-O operation occurs. No special action is required to release a record from its locked state if the record does not require any changes. If a requested record is already locked by another program, a file status of 9D is returned.

Unblocking Input Records and Blocking Output Records

To potentially improve the performance of input and output operations, the COBOL compiler generates code to unblock input records and block output records if all of the following conditions exist:

- ACCESS IS SEQUENTIAL is specified for the file.
- The file is opened only for INPUT or OUTPUT in that program.
- The file is assigned to DISK, DATABASE, DISKETTE, or TAPEFILE.
- No START statements are specified for the file.

Even when all of the above conditions are met, certain CPF restrictions can cause blocking and unblocking to not be performed. In these cases, performance improvements will not be realized.

The I-O-FEEDBACK area is not updated after each read or write for files in which multiple records are blocked and unblocked by COBOL. See "I-O-Feedback" in Appendix I for more information.

For data base files, you may not see all changes as they occur, if the changes are made in different programs. For a description of the effect of blocking on changes to data base files, see the discussion on sequential-only processing in the *CPF Programmer's Guide*.

Multiple Member Processing

Multiple member processing can be accomplished for a data base file, by overriding a data base file to process all members.

System Override Considerations

Any overrides must be specified before the file is opened by the COBOL program. The system uses the file override command to determine the file to open and the attributes of the file.

The simplest form of overriding a file is to override some attributes of the file. For example, FILE(OUTPUT) with COPIES(2) is specified when a printer file is created. Then, before the COBOL program is executed, the number of printed copies of output can be changed to 3. The override command is as follows:

```
OVRPRTF FILE(OUTPUT) COPIES(3)
```

Another form of file overriding is to redirect the COBOL program to access a different file. When the override redirects the program to a file of the same type (such as a printer file to another printer file), the file is processed in the same manner as the original file.

When the override redirects the program to a file of a different type, the overriding file is processed in the same manner as the original file would have been processed. However, device dependent specifications in the COBOL program are ignored and the defaults are taken by the system.

Not all file redirections are valid. A COBOL associated card file can only be overridden to another associated card file. An indexed file for a COBOL program can only be overridden to another indexed file with a keyed access path. A data base source file used for a COBOL program, cannot be overridden to process all members. Specifying OVRDBF MBR (*ALL) will result in the termination of the compile. A data base file used for a COPY statement, cannot be overridden to process all members. Specifying OVRDBF MBR (*ALL) will cause the COPY statement to be ignored.

It is the COBOL programmer's responsibility to ensure that file overrides are applied properly. For more information on valid file redirections, the device dependent characteristics ignored, and the defaults assumed, see the *CPF Programmer's Guide*.

Externally Described/Program Described Files

All files on System/38 are defined to CPF. However, the extent to which files can be defined differs:

- An *externally described file* is described at the field level to CPF through DDS. The description includes information about the type of file, such as data base or a device, and a description of each field and its attributes.
- A *program described file* is described at the field level within the COBOL program in the Data Division of the COBOL program. The description of the file to CPF includes information about the type of file and the length of the records in the file.

Both externally described files and program described files must be defined in the COBOL program within the INPUT-OUTPUT SECTION and the FILE SECTION. However, record descriptions in the FILE SECTION for externally described files can be defined with the DDS format of the COBOL COPY statement. (For more information on the COPY statement, DD or DDS format, see Chapter 6.)

Note: Actual file processing within the Procedure Division is the same, whether the file is externally described or program described.

Externally described files offer the following advantages:

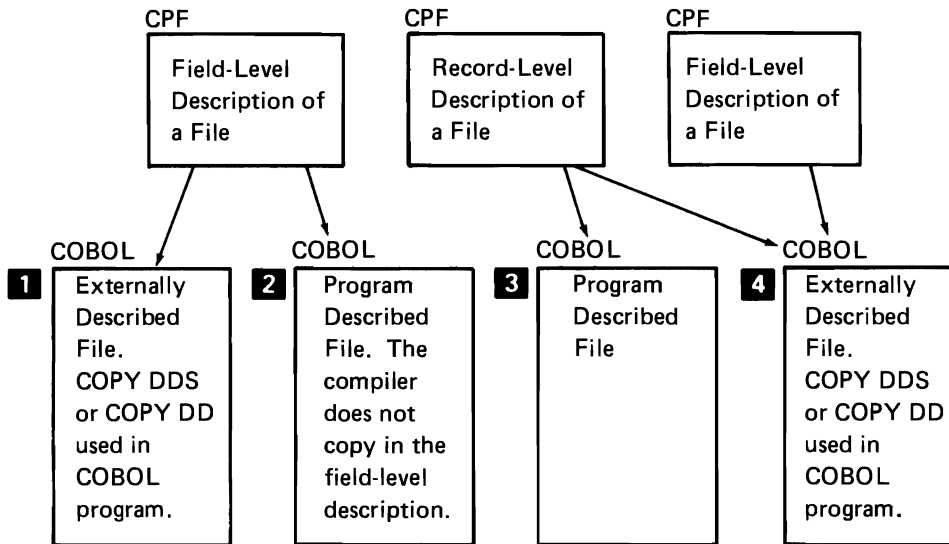
- Less coding in COBOL programs. If the same file is used by many programs, the fields can be defined once to CPF and used by all the programs. This eliminates the need to code record descriptions for COBOL programs that use externally described files.
- Less maintenance activity when the file's record format is changed. The user can often update programs by changing the file's record format and then recompiling the programs that use the files without changing any coding in the program.
- Improved documentation because programs using the same files use consistent record format and field names.

Device-dependent functions such as stacker selection or forms control are not extracted by the COPY DDS operation. Only field level descriptions are extracted. When EXTERNALLY-DESCRIBED-KEY is specified as RECORD KEY, the field(s) that compose RECORD KEY are also extracted from DDS.

If the user chooses, he can use an externally described file within the program by specifying the file as program described (specifying the coding for the record description in the source). In this case, the compiler does not copy in the external field-level description of the file at compilation time. This approach can be used

in conversion where existing programs use program described files and new programs use externally described files to refer to the same file.

Figure 9-1 shows some typical relationships between COBOL programs and files on System/38.



- 1** The COBOL program uses the field level description of a file that is defined to CPF. The COBOL user coded a COPY statement, DDS or DD format, for his record description. At compilation time, the compiler copies in the external field-level description and translates it into a syntactically correct COBOL record description. The file must exist at compilation time.
- 2** An externally described file is used as a program described file in the COBOL program. The entire record description for the file is coded in the COBOL program. This file does not have to exist at compilation time.
- 3** A file is described to CPF only to the record level. The entire record description must be coded in the COBOL program. This file does not have to exist at compilation time.
- 4** A file-name can be specified for compilation time, and a different file-name can be specified for execution time. A COBOL COPY statement, DDS or DD format, generates the record description for the file at compilation time. At execution time, a different library list or a file override command can be used so that a different file is accessed by the program. The file description copied in at compilation time is used to describe the input records used at execution time.

Note: For externally described files, the two file formats must be the same. Otherwise, a level check error will occur.

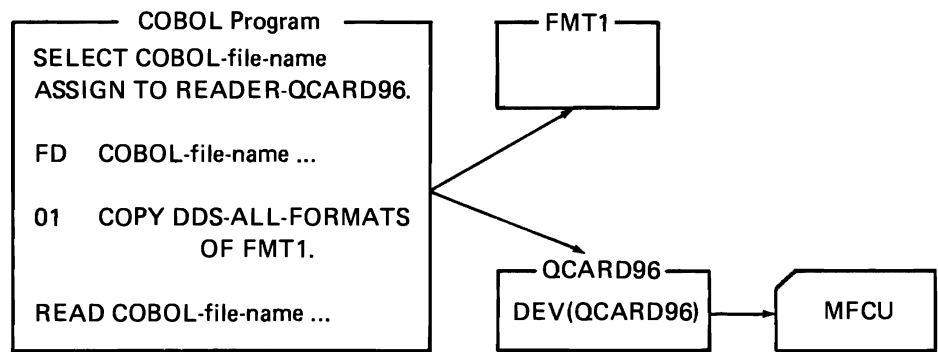
Figure 9-1. Typical Relationships between COBOL and the Files on System/38

By using the COBOL COPY statement, DDS or DD format, the user can generate a record description for a COBOL file from the record format for a different file. An override command is unnecessary. The following example shows an application where the file actually processed is the MFCU card device, QCARD96, but the record description for that file is generated from the record format for the file FMT1. To do this, the user:

- Defines a physical file named FMT1 that has one record format containing the description of each field in the record format. The record format is defined

on data description specifications (DDS). The externally described file should contain only one record format for a card device.

- Creates the file named FMT1 with a CPF Create Physical File command.
- Specifies the file-name of QCARD96 (the IBM-supplied device file name for the MFCU) in his COBOL program; uses the COBOL COPY statement, DDS format, to create the record description; and specifies READER as the device.
- Creates the COBOL program using the CRTCLPGM command.
- Calls the program at execution time. Because no file override is used at execution time for this program, the program then accesses the QCARD96 file:



Externally Described Files

For externally described files, COPY statements, DDS or DD format, are used for coding the record description for the file. Therefore, the file must be created before the program is compiled.

The external description for a file includes:

- The record format specifications that contain a description of the fields in a record
- Access path specifications that describe how the records are to be retrieved.

These specifications result from the DDS for the file and the CPF create file command that is used for the file.

Record Format Specifications

The record format specifications describe the fields in a record and the location of the fields in a record. The fields are located in the record in the order specified in DDS. The field description generally includes the field name, the field type (character, binary, zoned decimal, or packed decimal), and the field length (including the number of decimal positions in a numeric field). Instead of being specified in the record format for a physical or logical file, the field attributes can be defined in a field reference file (see Figure 9-2 and Figure 9-3).

The keys for a record format are specified in DDS. When you use a COPY statement, DDS or DD format, a table of comments is generated in the source program listing showing how the keys for the format are defined in DDS. The entries that may appear in the table and the table heading are listed below.

Heading	Possible Entry
NUMBER	key field number
NAME	key field name
RETRIEVAL TYPE	ASCENDING, DESCENDING ZONE, DIGIT, SIGNED, ABSVAL, AN (Alphanumeric), N (Numeric)
ALTSEQ	NO, YES

In addition, DDS keywords can be used to:

- Specify edit codes for a field (EDTCDE)
- Specify edit words for a field (EDTWRD)
- Specify that duplicate key values are not allowed for the file (UNIQUE)
- Specify a text description for a record format or a field (TEXT).

See the *CPF Reference Manual – DDS* for a complete list of the DDS keywords that are valid for a data base file.

Any editing that is to be performed on externally described output files is specified in DDS.

Figures 9-2 and 9-3 show an example of the DDS for a data base file and for a field reference file that defines the attributes for the fields used in the data base file. See the *CPF Programmer's Guide* for more information on a field reference file.

File		Keying Instruction		Graphic		Description		Page of				
Programmer		Date		Key								
A												
Sequence Number	Form Type	Conditioning					Name	Length	Reference (R)	Location		Functions
		And/Or/Comment (A/O/O*)	Indicator	Next (N)	Indicator	Next (N)				Line	Pos	
1												
	A					LST DAT		R				REFFLD(BASDAT)
	A											COLHDG('Last' 'Date' 'Paid')
7	A											TEXT('Last Date Paid in A/R')
	A					CRDLMT		R				REFFLD(ARBAL)
	A											COLHDG('Credit' 'Limit')
	A											TEXT('Customer Credit Limit')
	A					SLSYR		R+	2			REFFLD(ARBAL)
	A											COLHDG('Sales' 'This' 'Year')
	A											TEXT('Customer Sales This Year')
	A					SLSLYR		R+	2			REFFLD(ARBAL)
	A											COLHDG('Sales' 'Last' 'Year')
	A											TEXT('Customer Sales Last Year')

This example of a field reference file shows the definitions of the fields that are used by the CUSMSTL (customer master logical) file, which is shown in Figure 9-2. The field reference file normally contains the definitions of fields that are used by other files. The following text describes some of the entries for this field reference file.

- 1** The BASDAT field is edited by the Y edit code, as indicated by the keyword EDTCDE (Y). If this field is used in an externally described output file for a COBOL program, the COBOL-generated field is compatible with the data type specified in the DDS. The field is edited when the record is written. When the field is used in a program described output file, compatibility with the DDS fields in the file is the user's responsibility. When DDS is not used to create the file, appropriate editing of the field in the COBOL program is the user's responsibility.
- 2** The CHECK(MF) entry specifies that the field is a mandatory fill field when it is entered from a display work station. Mandatory fill means that all characters for the field must be entered from the display work station.
- 3** The ADDR and CITY fields share the same attributes that are specified for the NAME field, as indicated by the REFFLD keyword.
- 4** The RANGE keyword, which is specified for the CUSTYP field, ensures that the only valid numbers that can be entered into this field from a display work station are 1 through 5.
- 5** The COLHDG keyword provides a column head for the field if it is used by the Interactive Data Base Utilities (IDU).
- 6** The ARBAL field is edited by the J edit code, as indicated by the keyword EDTCDE(J).
- 7** A text description (TEXT keyword) is provided for some fields. The TEXT keyword is used for documentation purposes and appears in various listings.

Figure 9-3 (Part 2 of 2). Example of a Field Reference File

Access Path

The description of an externally described file contains the access path that describes how records are to be retrieved from the file. Records can be retrieved based on an arrival sequence (nonkeyed) access path or on a keyed sequence access path.

The arrival sequence access path is based on the order in which the records are stored in the file. Records are added only to the end of the file.

For the keyed sequence access path, the sequence in which records are retrieved from the file is based on the contents of the key field(s) that is defined in the DDS for the file. For example, in the DDS shown in Figure 9-2, CUST is defined as the key field. The keyed sequence access path is updated whenever records are added, deleted, or the contents of a key field change.

See the *CPF Programmer's Guide* for a complete description of the access paths for an externally described data base file.

Record Keys and Common Keys

For a keyed sequence access path, one or more fields can be defined in the DDS to be used as the key fields for a record format. All record types in a file do not have to have the same key fields. For example, an order header record can have the ORDER field defined as the key field, and the order detail records can have the ORDER and LINE fields defined as the key fields.

The key for a file is determined by the valid keys for the record types in that file. The file's key is determined in the following manner:

- If all record types in a file have the same number of key fields defined in DDS that are identical in attributes, the *key for the file* consists of all fields in the key for the record types. (The corresponding fields do not have to have the same name.) For example, if the file has three record types and the key for each record type consists of fields A, B, and C, then the file's key consists of fields A, B, and C. That is, the file's key is the same as the records' key.
- If all record types in the file do not have the same key fields, the key for the file consists of the key fields *common* to all record types. For example, a file has three record types and the key fields are defined as follows:
 - REC1 contains key field A.
 - REC2 contains key fields A and B.
 - REC3 contains key fields A, B, and C.

Then the file's key is field A, the key field common to all record types.

- If no key field is common to all record types, any keyed reference to the file will always return the first record in the file.

In COBOL you must specify a RECORD KEY for an indexed file to identify the record you want to process. COBOL compares the key value with the key of the file or record, and performs the specified operation on the record whose key matches the RECORD KEY value.

When RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified:

- If the FORMAT phrase is specified, the compiler builds the search argument from the key fields in the record area for the specified format
- If the FORMAT phrase is not specified, the compiler builds the search argument from the key fields in the record area for the first record format defined in the program for that file.

Programming Note: For a file containing multiple key fields to be processed in COBOL, the key fields must be contiguous in the record format used by the COBOL program, except when RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified.

COBOL Specifications for Externally Described Files

Data description specifications (DDS) are used to describe files at the field level to CPF. Each record format in an externally described file is identified by a unique record format name. Figure 9-4 shows an example of the DDS for a file with one record format (one record type).

The COBOL user can code a COPY statement, DDS format, of one of the following forms to have the external description retrieved:

```
COPY DDS-CUSREC-I-O OF CUSMSTL .
```

```
COPY DDS-CUSREC OF CUSMSTL .
```

```
COPY DD-CUSREC OF CUSMSTL .
```

The information from the external description is then retrieved by the COBOL compiler, and a COBOL data structure is generated. Figure 9-5 shows the COBOL code generated by the COPY statement, DDS format.

Figure 9-6 shows an example of DDS with the Alias keyword. Figure 9-7 shows the COBOL code generated by the COPY statement, DD format.

Actual file processing within the Procedure Division is the same for both program described and externally described files.


```

01 CUS-MASTER.
   COPY DDS-CUSREC OF CUSTMAST-CUSLIB.
*I-O FORMAT: CUSREC FROM FILE CUSTMAST OF LIBRARY CUSLIB          CUSREC
*                                                                    CUSREC
*THE KEY DEFINITIONS FOR THE RECORD FORMAT CUSREC
*NUMBER  NAME      RETRIEVAL  TYPE    ALTSEQ
*0001    CUST      ASCENDING  AN      NO
   05     CUSREC.
*         06 CUST      PIC X(5).          CUSREC
*         CUSTOMER NUMBER          CUSREC
*         06 NAME      PIC X(20).         CUSREC
*         CUSTOMER NAME&           CUSREC
*         06 ADDR      PIC X(20).         CUSREC
*         CUSTOMER ADDRESS         CUSREC
*         06 CITY      PIC X(20).         CUSREC
*         CUSTOMER CITY            CUSREC
*         06 STATE     PIC X(2).          CUSREC
*         STATE ABBREVIATION       CUSREC
*         06 ZIP       PIC S9(5)         COMP-3. CUSREC
*         ZIP CODE                  CUSREC
*         06 SHRCOD   PIC X(3).          CUSREC
*         CUSTOMER NAME SEARCH CODE CUSREC
*         06 CUSTYP   PIC X(1).          CUSREC
*         CUSTOMER TYPE             CUSREC
*         06 ARBAL    PIC S9(8)V9(2)     COMP-3. CUSREC
*         ACCT/REC BALANCE          CUSREC

```

Figure 9-5. Example of the Results of the COPY Statement, DDS Format

Overriding or Adding COBOL Functions to the External Description

In addition to placing the external file description in the program through the use of the COPY statement, DDS format, the user can also use standard record definition and redefinition to describe external files or to provide a group definition for a series of fields. It is the programmer's responsibility to ensure that program described definitions are compatible with the external definitions of the file.

File	Date	Keying Instruction	Graphic					
Programmer			Key					

Description	Page	of
-------------	------	----

Sequence Number	Form Type (A/O/*)	Conditioning				Name	Length	Reference (R)	Data Type (B A/P/S/B A/S/X/Y/N/M)	Decimal Positions	Usage (B/O//B/H/M)	Location		Functions
		Indicator	Not (N)	Indicator	Not (N)							Line	Pos	
A														
					LOGICAL	CUSMSTL								CUSTOMER MASTER FILE
A														UNIQUE
A					R	CUSREC								PF FILE(CUSMSTP)
A														TEXT('CUSTOMER MASTER RECORD')
A						CUST								1 ALIAS(CUSTOMER_NUMBER)
A						NAME								ALIAS(CUSTOMER_NAME)
A						ADDR								ALIAS(ADDRESS)
A						CITY								
A						STATE								
A						ZIP								
A						SRHCOD								ALIAS(SEARCH_CODE)
A						CUSTYP								ALIAS(CUSTOMER_TYPE)
A						ARBAL								ALIAS(ACCT_REC_BALANCE)
A						ORDBAL								
A						LSTAMT								
A						LSTDAT								
A						CRDLMT								
A						SLSYR								
A						SLSLYR								
A					K	CUST								

*Number of sheets per pad may vary slightly.

1 The name associated with the Alias keyword, which will be included in the program.

Figure 9-6. Example of Data Description Specifications with Alias

```

01 CUS-MASTER.
   COPY DD-CUSREC OF CUSTMAST-CUSLIB.
*I-O FORMAT: CUSREC FROM FILE CUSTMAST OF LIBRARY CUSLIB      CUSREC
*
*THE KEY DEFINITIONS FOR THE RECORD FORMAT CUSREC
*NUMBER  NAME                RETRIEVAL TYPE                ALTSEQ
*0001    CUSTOMER-NUMBER      ASCENDING  AN                    NO

   05 CUSREC.
      06 CUSTOMER-NUMBER      PIC X(5).                CUSREC
      *                      CUSTOMER NUMBER                CUSREC
      06 CUSTOMER-NAME        PIC X(20).                CUSREC
      *                      CUSTOMER NAME                    CUSREC
      06 ADDRESS               PIC X(20).                CUSREC
      *                      CUSTOMER ADDRESS                CUSREC
      06 CITY                  PIC X(20).                CUSREC
      *                      CUSTOMER CITY                    CUSREC
      06 STATE                 PIC X(2).                CUSREC
      *                      STATE ABBREVIATION              CUSREC
      06 ZIP                   PIC S9(5) COMP-3.          CUSREC
      *                      ZIP CODE                          CUSREC
      06 SEARCH-CODE           PIC X(3).                CUSREC
      *                      CUSTOMER NAME SEARCH CODE      CUSREC
      06 CUSTOMER-TYPE         PIC X(1).                CUSREC
      *                      CUSTOMER TYPE                    CUSREC
      06 ACCT-REC-BALANCE      PIC S9(8)V9(2) COMP-3.    CUSREC
      *                      ACCT/REC BALANCE                CUSREC

```

Figure 9-7. Example of the Results of the Copy Statement, DD Format with the Alias Keyword

Program Described Files

Records and fields for a program described file are described by coding record descriptions in the File Section of the COBOL program instead of using the COPY statement, DDS or DD format.

The file must be created on the system before the program can be executed. This can be done by using one of the Create File commands. See the *CL Reference Manual* for the commands.

DDS can be used with the Create File commands. For a COBOL indexed file, a keyed access path must be created. This can be done by specifying a key in DDS when the file is created. The record key in COBOL must match the key defined when the file was created.

Specific COBOL File Processing

Printer File Considerations

You can obtain printed output from a COBOL program by issuing WRITE statements to one or more printer files. Each printer file must have a unique name and be assigned to a device of PRINTER or FORMATFILE in the ASSIGN clause of that file's FILE-CONTROL entry. A device of PRINTER must be used for program described files, and a device of FORMATFILE must be used for externally described printer files. The Create Print File (CRTPRTF) command can be used to create a printer file (see the *CL Reference Manual* for further information on the CRTPRTF command), or one of the IBM-supplied printer device files, such as QPRINT can be used.

The file operations that are valid for a printer file are WRITE, OPEN, and CLOSE. For a complete description of these operations, see Chapter 5.

FORMATFILE must be used when the file is an externally described printer file. See the *CPF Programmer's Guide* for information on the DDS for externally described printer files.

SPECIAL-NAMES Paragraph and the ADVANCING Phrase

When the mnemonic-name associated with the function-name CSP is specified in the ADVANCING phrase of a WRITE statement for a printer file, it has the same effect as specifying ADVANCING 0 LINES.

When the mnemonic-name associated with the function-name C01 is specified in the ADVANCING phrase of a WRITE statement for a printer file, it has the same effect as specifying ADVANCING PAGE.

The ADVANCING phrase cannot be specified in WRITE statements for files assigned to FORMATFILE.

LINAGE Clause

When the LINAGE clause is specified for a file assigned to PRINTER, all spacing and paging controls are handled internally by the compiler. At OPEN time, the printer is positioned to a new physical page and the LINAGE-COUNTER is set to 1. All spacing or paging for following WRITE statements for the file is controlled internally, and the physical page size is ignored. For a file that has a LINAGE clause and is assigned to PRINTER, paging consists of spacing to the end of the logical page (page body) and then spacing past the bottom and top margins.

Use of the LINAGE clause degrades performance. The LINAGE clause should be used only as necessary. If the physical paging is acceptable, the LINAGE clause is not necessary.

The LINAGE clause should not be used for files assigned to FORMATFILE.

FORMATFILE Files

Externally described printer files must be assigned to a device of FORMATFILE. The term FORMATFILE is used because the FORMAT phrase is valid in WRITE statements for the file, and the data formatting is specified in the DDS for the file.

When you have specified a device of FORMATFILE, you can obtain formatting of printed output in two ways:

1. Choose which formats to print in which order by using appropriate values in the FORMAT phrases specified for WRITE statements. For example, use one format once per page to produce a heading, and use another format to produce the detail lines on the page.
2. Choose the appropriate options to be taken when each format is printed by setting indicator values and passing these indicators through the INDICATOR phrase for the WRITE statement. For example, fields may be underlined, blank lines may be produced before or after the format is printed, or the printing of certain fields may be skipped.

The use of external descriptions for printer files has the following advantages over program descriptions:

- Multiple lines can be printed by one WRITE statement. When multiple lines are written by one WRITE statement and the END-OF-PAGE condition is reached, the END-OF-PAGE imperative statement is executed after all of the lines are printed. It is possible to print lines in the overflow area, and onto the next page before the END-OF-PAGE imperative statement is executed.

Figure 9-8 shows an example of an occurrence of the END-OF-PAGE condition through FORMATFILE.

- Optional printing of fields based on indicator values is possible.
- Editing of field values is easily defined.
- Maintenance of print formats, especially those used by multiple programs, is easier.

Use of the ADVANCING phrase for FORMATFILE files causes a compilation error to be issued. Advancing of lines is controlled in a FORMATFILE file through DDS keywords such as SKIPA and SKIPB, and through the use of line numbers.

For FORMATFILE files, the LINAGE clause is invalid.

Figure 9-9 shows an example of using an externally described printer file through FORMATFILE.

```

                                COBOL SOURCE LISTING
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME  CHG/DATE

1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID. FORMATFILE.
3 000300 AUTHOR. IBM CANADA LABORATORY.
4 000400 ENVIRONMENT DIVISION.
5 000500 CONFIGURATION SECTION.
6 000600 SOURCE-COMPUTER. IBM-S38.
7 000700 OBJECT-COMPUTER. IBM-S38.
8 000800 INPUT-OUTPUT SECTION.
9 000900 FILE-CONTROL.
10 001000 SELECT PERSREPT ASSIGN TO FORMATFILE-PERSREPT-SI
11 001100 ORGANIZATION IS SEQUENTIAL.
12 001200 SELECT PERSFILE ASSIGN TO DATABASE-PERSFILE
13 001300 ORGANIZATION IS INDEXED
14 001400 ACCESS MODE IS SEQUENTIAL
15 001500 RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.
16 001600 DATA DIVISION.
17 001700 FILE SECTION.
18 001800
19 001900 FD PERSREPT
20 002000 LABEL RECORDS ARE STANDARD.
21 002100 01 PERSREPT-REC.
22 002200 COPY DDS-ALL-FORMATS-O OF PERSREPT-EXAMPLES.
23 +000001 05 PERSREPT-RECORD PIC X(146).
24 +000002* OUTPUT FORMAT:HEADING FROM FILE PERSREPT OF LIBRARY EXAMPLES <-ALL-FMTS
+000003* <-ALL-FMTS
24 +000004 05 HEADING-O REDEFINES PERSREPT-RECORD. <-ALL-FMTS
25 +000005 06 ORDERTYPE PIC X(15). <-ALL-FMTS
+000006* OUTPUT FORMAT:DETAIL FROM FILE PERSREPT OF LIBRARY EXAMPLES <-ALL-FMTS
+000007* <-ALL-FMTS
26 +000008 05 DETAIL-O REDEFINES PERSREPT-RECORD. <-ALL-FMTS
27 +000009 06 NAME PIC X(30). <-ALL-FMTS
28 +000010 06 EEMPLNO PIC S9(6). <-ALL-FMTS
29 +000011 06 BIRTHDATE PIC X(6). <-ALL-FMTS
30 +000012 06 ADDRESS1 PIC X(35). <-ALL-FMTS
31 +000013 06 MARSTAT PIC X(11). <-ALL-FMTS
32 +000014 06 SPOUSENAME PIC X(30). <-ALL-FMTS
33 +000015 06 ADDRESS2 PIC X(35). <-ALL-FMTS
34 +000016 06 NUMCHILD PIC S9(2). <-ALL-FMTS

```

Figure 9-8 (Part 1 of 3). Example of the END-OF-PAGE Condition

```

                                COBOL SOURCE LISTING          FORMATFILE
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME  CHG/DATE

002300/
35 002400 FD PERSFILE
36 002500 LABEL RECORDS ARE STANDARD.
37 002600 01 PERSFILE-REC.
38 002700 COPY DDS-ALL-FORMATS OF PERSFILE-EXAMPLES.
39 +000001 05 PERSFILE-RECORD PIC X(145).
+000002* I-O FORMAT:PERSREC FROM FILE PERSFILE OF LIBRARY EXAMPLES <-ALL-FMTS
+000003* <-ALL-FMTS
+000004*THE KEY DEFINITIONS FOR RECORD FORMAT PERSREC RETRIEVAL TYPE ALTSEO <-ALL-FMTS
+000005* NUMBER NAME ASCENDING SIGNED NO <-ALL-FMTS
40 +000006* 0001 EEMPLNO REDEFINES PERSFILE-RECORD. <-ALL-FMTS
41 +000007 05 PERSREC <-ALL-FMTS
42 +000008 06 EEMPLNO PIC S9(6). <-ALL-FMTS
43 +000009 06 NAME PIC X(30). <-ALL-FMTS
44 +000010 06 ADDRESS1 PIC X(35). <-ALL-FMTS
45 +000011 06 ADDRESS2 PIC X(35). <-ALL-FMTS
46 +000012 06 BIRTHDATE PIC X(6). <-ALL-FMTS
47 +000013 06 MARSTAT PIC X(11). <-ALL-FMTS
48 +000014 06 SPOUSENAME PIC X(30). <-ALL-FMTS
49 +000015 06 NUMCHILD PIC S9(2). <-ALL-FMTS
49 002800
50 002900 WORKING-STORAGE SECTION.
51 003000 77 HEAD-ORDER PIC X(15) VALUE "EMPLOYEE NUMBER".
52 003100 01 PERSREPT-INDICS.
53 003200 COPY DDS-ALL-FORMATS-O-INDIC OF PERSREPT-EXAMPLES.
54 +000001 05 PERSREPT-RECORD.
+000002* OUTPUT FORMAT:HEADING FROM FILE PERSREPT OF LIBRARY EXAMPLES <-ALL-FMTS
+000003* <-ALL-FMTS
+000004* 06 HEADING-O-INDIC. <-ALL-FMTS
+000005* OUTPUT FORMAT:DETAIL FROM FILE PERSREPT OF LIBRARY EXAMPLES <-ALL-FMTS
+000006* <-ALL-FMTS
55 +000007 06 DETAIL-O-INDIC. <-ALL-FMTS
56 +000008 07 IND1 PIC 1 INDIC 01. <-ALL-FMTS
57 003300
58 003400 77 EOF-FLAG PIC X VALUE "0".
59 003500 88 NOT-END-OF-FILE VALUE "0".
60 003600 88 END-OF-FILE VALUE "1".
61 003700 77 MARRIED PIC X VALUE "M".

```

Figure 9-8 (Part 2 of 3). Example of the END-OF-PAGE Condition

```

                                COBOL SOURCE LISTING          FORMATFILE
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S COPYNAME  CHG/DATE

003800/
62 003900 PROCEDURE DIVISION.
004000 FIRST-SECT SECTION.
004100 FIRST-PARA.
63 004200 OPEN INPUT PERSFILE
004300 OUTPUT PERSREPT.
64 004400 PERFORM HEADING-LINE.
65 004500 PERFORM PROCESS-RECORD UNTIL END-OF-FILE.
66 004600 CLOSE PERSFILE PERSREPT.
67 004700 STOP RUN.
004800
004900 PROCESS-RECORD.
68 005000 READ PERSFILE AT END SET END-OF-FILE TO TRUE.
70 005100 IF NOT-END-OF-FILE THEN
71 005200 PERFORM PRINT-RECORD.
005300
005400 PRINT-RECORD.
72 005500 CLOSE PERSFILE PERSREPT.
73 005600 MOVE CORR PERSREC TO DETAIL-O.
74 005700 IF MARSTAT IN PERSFILE-REC = MARRIED THEN
75 005800 MOVE B*1" TO INO1 IN DETAIL-O-INDIC
005900 ELSE
76 006000 MOVE B*0" TO INO1 IN DETAIL-O-INDIC.
77 006100 WRITE PERSREPT-REC FORMAT IS "DETAIL"
006200 INDICATORS ARE DETAIL-O-INDIC
78 006300 AT EOP PERFORM HEADING-LINE. 1
006400 HEADING-LINE.
79 006500 MOVE HEAD-ORDER TO ORDERTYPE.
80 006600 WRITE PERSREPT-REC FORMAT IS "HEADING".

***** END OF SOURCE *****

```

1 If the number of lines per page has been exceeded, END-OF-PAGE occurs. The format HEADING is printed on a new page.

Figure 9-8 (Part 3 of 3). Example of the END-OF-PAGE Condition

File		Keying Instruction	Graphic				Description		Page of	
Programmer	Date		Key							

Sequence Number	Form Type And/Or Comment (A/O/*) Not (N)	Conditioning					Name	Length	Reference (R)	Data Type (B/A/P/S/B/F/A/S/X/Y/N/I/M) Decimal Positions Usage (B/O/I/B/H/M/P)	Location		Functions	
		Indicator	Indicator	Indicator	Indicator	Indicator					Line	Pos		
1	A*						PRINTER FILE DDS FOR							
2	A*						FORMATFILE EXAMPLE							
3	A												1 INDARA REF (PERFILE)	
4	A						R HEADING	2					SKIP B(1) SPACEA(3)	
5	A												15 PERSONNEL LISTING	
6	A												UNDERLINE	
7	A												33 BORDERED BY	
8	A						ORDERTYPE	13					36	
9	A												80 DATE EDTCDE(Y)	
10	A												93 TIME	
11	A												115 PAGE: 1	
12	A												+1 PAGE NR EDTCDE(3)	
13	A*						R DETAIL	5					SPACEA(3)	
14	A*						LINE 1							
15	A												1 NAME: 1	
16	A						NAME	R					11 UNDERLINE	
17	A												55 EMPLOYEE NUMBER: 1	
18	A						EMPLNO	R					73	
19	A												87 DATE OF BIRTH: 1	
20	A						BIRTHDATE	R					103 SPACEA(1)	
21	A*						LINE 2							
22	A												1 ADDRESS: 1	
23	A						ADDRESS1	R					11	
24	A	01											55 MARITAL STATUS: 1	
25	A	01					MARSTAT	R					73	
26	A*						LINE 3							
27	A						SPOUSE NAME							87 SPOUSE'S NAME: 1
28	A												103	
29	A						ADDRESS2	R					11 SPACEB(1)	
30	A												55 CHILDREN: 1	
31	A						NINCHILD	R					73 EDTCDE(3)	
32	A													

*Number of sheets per pad may vary slightly.

Figure 9-9 (Part 2 of 7). Example of the Use of Externally Described Printer Files Assigned to a Device of FORMATFILE

- 1** INDARA specifies that a separate indicator area is to be used for the file.
- 2** HEADING is the format name which provides headings for each page.
- 3** SKIPB(1) and SPACEA(3) are used to:
 1. Skip to line 1 of the next page before format HEADING is printed.
 2. Leave 3 blank lines after format HEADING is printed.
- 4** DATE, TIME and PAGNBR are used to have the current date, time and page number printed automatically when format HEADING is printed.
- 5** DETAIL is the format name used to print the detail line for each employee in the personnel file.
- 6** SPACEA(3) causes 3 lines to be left blank after each employee detail line.
- 7** SPACEA(1) causes a blank line to be printed after the field BIRTHDATE is printed. As a result, subsequent fields in the same format are printed on a new line.
- 8** 01 means that these fields are printed only if the COBOL program turns indicator 01 on and passes it when format DETAIL is printed.
- 9** EDTCDE(3) is used to remove leading zeros when printing this numeric field.

Figure 9-9 (Part 3 of 7). Example of the Use of Externally Described Printer Files Assigned to a Device of FORMATFILE

```

5714C91 R05 M00 830610          COBOL SOURCE LISTING          03/04/93
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN 5  C.PYNAME

1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID, FORMATFILE.
3 000300 AUTHOR, IBM CANADA LABORATORY.
4 000400 ENVIRONMENT DIVISION.
5 000500 CONFIGURATION SECTION.
6 000600 SOURCE-COMPUTER, IBM-S39.
7 000700 OBJECT-COMPUTER, IBM-S39.
8 000800 INPUT-OUTPUT SECTION.
9 000900 FILE-CONTROL.
10 001000     SELECT PERSREPT ASSIGN TO FORMATFILE-PERSR=PT-SI
11 001100         ORGANIZATION IS SEQUENTIAL.
12 001200     SELECT PERSFILE ASSIGN TO DATABASE-PERSFILE
13 001300         ORGANIZATION IS INDEXED
14 001400         ACCESS MODE IS SEQUENTIAL
15 001500         RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.
16 001600 DATA DIVISION.
17 001700 FILE SECTION.
18 001800
19 001900 FD PERSREPT
20 002000     LABEL RECORDS ARE STANDARD.
21 002100     01 PERSREPT-REC.
22 002200     COPY DDS-ALL-FORMATS-D OF PERSREPT-EXAMPLES.
23 002300     05 PERSREPT-RECORD PIC X(146).
24 +000001
25 +000002
+000003* OUTPUT FORMAT:HEADING      FROM FILE PERSREPT  OF LIBRARY EXAMPLES
+000004*
26 +000005     05 HEADING-D      REDEFINES PERSREPT-RECORD.
27 +000006     06 HDERTYPE      PIC X(15).
28 +000007
+000008* OUTPUT FORMAT:DETAIL      FROM FILE PERSREPT  OF LIBRARY EXAMPLES
+000009*
29 +000010     05 DETAIL-D      REDEFINES PERSREPT-RECORD.
30 +000011     06 NAME          PIC X(30).
31 +000012     06 EEMPLND      PIC $9(6).
32 +000013     06 BIRTHDATE    PIC X(6).
33 +000014     05 ADDRESS1     PIC X(35).
34 +000015     06 MARSTAT      PIC X(1).
35 +000016     05 SPOUSENAME    PIC X(30).
36 +000017     06 ADDRESS2     PIC X(35).
37 +000018     05 NUMCHLD      PIC $9(2).

```

Figure 9-9 (Part 4 of 7). Example of the Use of Externally Described Printer Files Assigned to a Device of FORMATFILE

```

5714CB1 R05 M00 830610          COBOL SOURCE LISTING          FORMATFILE          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME

002300/
38 002400 FD PERSFILE
39 002500 LABEL RECORDS ARE STANDARD.
40 002600 01 PERSFILE-REC.
41 002700 COPY DDS-ALL-FORMATS OF PERSFILE-EXAMPLES.
43 +000001 05 PERSFILE-RECORD PIC X(145). <-ALL-FMTS
44 +000002 <-ALL-FMTS
+000003 I-O FORMAT:PERSREC FROM FILE PERSFILE OF LIBRARY EXAMPLES <-ALL-FMTS
+000004 <-ALL-FMTS
+000005 THE KEY DEFINITIONS FOR RECORD FORMAT PERSREC <-ALL-FMTS
+000006 NUMBER NAME RETRIIVAL TYPE ALTSEQ <-ALL-FMTS
+000007 0001 FPLND ASCENDING N NO <-ALL-FMTS
45 +000008 05 PERSREC REDEFINES PERSFILE-RECORD. <-ALL-FMTS
46 +000009 06 EEMPLND PIC 5(16). <-ALL-FMTS
47 +000010 06 NAME PIC X(30). <-ALL-FMTS
48 +000011 06 ADDRESS1 PIC X(35). <-ALL-FMTS
49 +000012 06 ADDRESS2 PIC X(35). <-ALL-FMTS
50 +000013 06 BIRTHDATE PIC X(6). <-ALL-FMTS
51 +000014 06 MARSTAT PIC X(1). <-ALL-FMTS
52 +000015 06 SPOUSENAME PIC X(30). <-ALL-FMTS
53 +000016 06 NUMCHILD PIC 5(12). <-ALL-FMTS
54 002800
55 002900 WORKING-STORAGE SECTION.
56 003000 01 PERSREPT-FORMAT-INFO.
57 003100 05 HEAD-LINES PIC 9 COMP VALUE 3.
58 003200 05 HEAD-ORDER PIC X(15) VALUE "EMPLOYEE NUMBER".
59 003300 05 DETAIL-LINES PIC 9 COMP VALUE 5.
60 003400
61 003500 01 PERSREPT-INDICS.
62 003600 COPY DDS-ALL-FORMATS-O-INDIC OF PERSREPT-EXAMPLES. 4
64 +000001 05 PERSREPT-RECORD. <-ALL-FMTS
65 +000002 <-ALL-FMTS
+000003 OUTPUT FORMAT:HEADING FROM FILE PERSREPT OF LIBRARY EXAMPLES <-ALL-FMTS
+000004 <-ALL-FMTS
+000005 06 HEADING-O-INDIC. <-ALL-FMTS
66 +000006 <-ALL-FMTS
+000007 OUTPUT FORMAT:DETAIL FROM FILE PERSREPT OF LIBRARY EXAMPLES <-ALL-FMTS
+000008 <-ALL-FMTS
67 +000009 06 DETAIL-O-INDIC. <-ALL-FMTS
68 +000010 07 IN01 PIC 1 INDIC 01. <-ALL-FMTS
69 003700
70 003800 77 EOF-FLAG PIC X VALUE "0".
71 003900 88 NOT-END-OF-FILE VALUE "0".
72 004000 89 END-OF-FILE VALUE "1".
73 004100 77 MARRIED PIC X VALUE "M".
74 004200 77 LINE-COUNT PIC 9(3) COMP VALUE 99.
75 004300 77 PAGE-SIZE PIC 9(3) COMP VALUE 55.

```

Figure 9-9 (Part 5 of 7). Example of the Use of Externally Described Printer Files Assigned to a Device of FORMATFILE

```

5714CB1 R05 M00 830610          COBOL SOURCE LISTING          FORMATFILE          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME

004400/
76 004500 PROCEDURE DIVISION.
004600 FIRST-SECT SECTION.
004700 FIRST-PARA.
004800 OPEN INPUT PERSFILE
004900 OUTPUT PERSREPT.
78 005000 PERFORM PROCESS-RECORD UNTIL END-OF-FILE.
79 005100 CLOSE PERSFILE PERSREPT.
80 005200 STOP RUN.
005300
005400 PROCESS-RECORD.
81 005500 READ PERSFILE AT END SET END-OF-FILE TO TRUE.
83 005600 IF NOT-END-OF-FILE THEN
84 005700 PERFORM PRINT-RECORD. 5
005800
005900 PRINT-RECORD.
85 006000 IF LINE-COUNT > PAGE-SIZE THEN 6
86 006100 MOVE HEAD-ORDER TO ORDERTYPE
87 006200 WRITE PERSREPT-REC FORMAT IS "HEADING"
88 006300 MOVE HEAD-LINES TO LINE-COUNT.
006400
89 006500 7 MOVE CURR PERSREC TO DETAIL-O.
90 006600 IF MARSTAT IN PERSFILE-REC = "MARRIED THEN 8
91 006700 MOVE B"1" TO IN01 IN DETAIL-O-INDIC 8
006800 ELSE
92 006900 MOVE B"0" TO IN01 IN DETAIL-O-INDIC. 9
93 007000 WRITE PERSREPT-REC FORMAT IS "DETAIL" 9
007100 INDICATORS ARE DETAIL-O-INDIC.
94 007200 ADD DETAIL-LINES TO LINE-COUNT. 10

***** END OF SOURCE *****

```

Figure 9-9 (Part 6 of 7). Example of the Use of Externally Described Printer Files Assigned to a Device of FORMATFILE

- 1** The externally described printer file is assigned to device `FORMATFILE`.
- 2** The `COPY` statement, `DDS` format, is used to copy the fields for the printer file into the program.
- 3** Note that although the fields in format `DETAIL` will be printed on 3 separate lines, they are defined in one record.
- 4** `COPY-DDS` is used to copy the indicators used in the printer file into the program.
- 5** Paragraph `PROCESS-RECORD` performs `PRINT-RECORD` for each employee record.
- 6** If the number of lines per page has been exceeded, format `HEADING` is printed on a new page. The current line count is reset to the number of lines taken up by `HEADING`.
- 7** All fields in the employee record are moved to the record for format `DETAIL`.
- 8** If the employee is married, indicator `01` is turned on; otherwise the indicator is turned off, preventing the spouse's name field in `DETAIL` from being printed.
- 9** Format `DETAIL` is printed with indicator `01` passed to control printing.
- 10** The number of lines (including blank lines) taken up by `DETAIL` is added to the current line count.

Figure 9-9 (Part 7 of 7). Example of the Use of Externally Described Printer Files Assigned to a Device of `FORMATFILE`

Card File Considerations

The following paragraphs discuss the COBOL processing unique to card files. This processing includes card file function specification, hopper selection, stacker selection, nonassociated card files, and associated card files.

Specifying the Function

For System/38 COBOL, card file processing is divided into four I-O functions: reading, punching, printing, and punching-printing. The device-name specified in the `ASSIGN` clause for a card file is a function selection. The valid devices are `READER`, `PUNCH`, `PRINT`, and `PUNCHPRINT`. At execution time, the system-name specified in the `ASSIGN` clause must correspond to a card file specified on a `Create Card File (CRTCRDF)` command.

Nonassociated Card Files

Card files are nonassociated if no association-number is specified in the `ASSIGN` clause. Nonassociated card file processing is limited to processing of only one of the four I-O functions. Thus, only one nonassociated card file can be open at a time, and only its specified function can be performed.

Associated Card File

Card files are associated when an association-number is specified in the ASSIGN clause. All card files with the same association-number are then processed at the same time. They must all be open before any processing is done, and all processing must be complete when a CLOSE is issued for any of the files. Associated card file processing allows the functions of reading, punching, and printing to be combined for one system and file. Only one set of associated card files per MFCU can be processed (open) at a time. Associated card files cannot be spooled.

File status on associated files is set only on the COBOL verb that actually causes the 5424 Multi-Function Card Unit (MFCU) to process the card. For example, if a READER and PUNCH or PRINT or PUNCHPRINT are associated, the card is processed only on the READ statement execution, and file status is set only for the READ statement. If a PUNCH and PRINT are associated, and the READER is not specified, card processing occurs only on the WRITE (Print) verb, and file status is set only for the WRITE statement. See Appendix D for more information on associated card file processing.

Stacker Selection

Stacker selection allows the user to specify the stacker that the cards are to be put into after a punch or print operation. The stacker selection is done in the COBOL program by:

1. Specifying a mnemonic-name for the function-names S01, S02, S03, S04, or S05 in the SPECIAL-NAMES paragraph. S01 through S04 correspond to stackers 1 through 4, and S05 corresponds to stacker 1.
2. Specifying the mnemonic-name for the desired stacker in the WRITE ADVANCING mnemonic-name statement for the card file.

If a stacker is not selected by the program, stacker 1 is assumed.

Hopper Selection

Hopper selection allows the user to select the specific hopper from which the cards are to be processed. The hopper is selected in the ASSIGN clause by specifying P for the primary hopper and S for the secondary hopper. If a hopper is not specified, the primary hopper is assumed. The primary hopper must be used for associated card files.

A hopper can also be specified on the CRTCRDF command. For information on the CRTCRDF command, see the *CL Reference Manual*.

DISK and DATABASE File Considerations

Data base files, which are associated with the COBOL devices of DATABASE and DISK, can be:

- Externally described files, whose fields are described to CPF through DDS
- Program described files, whose fields are described in the program that uses the file.

All data base files are created by CPF create file commands. See the *CPF Programmer's Guide* for a description of the CPF commands that relate to data base files.

DATABASE versus DISK Files

Assigning a file to DISK in COBOL restricts the user to traditional DISK processing. The use of DATABASE as the device permits the user to make use of the special System/38 COBOL data base features such as formats and duplicate record keys.

Processing Methods for DISK and DATABASE Files

COBOL Indexed Files: An indexed file is a file whose access path is built on key values. The user must create a keyed access path for an indexed file by using DDS.

To write standard ANS COBOL X3.23-1974 to access an indexed file, the file must be created with certain characteristics. The following table lists these characteristics and what controls them.

Characteristic	Control
The file must be a physical file.	The CRTPF CL command
The file cannot have records with duplicate key values.	The DDS keyword UNIQUE
The file cannot be a shared file.	The CRTPF CL command
A key must be defined for the file.	DDS
Keys must be in ascending sequence.	DDS
Keys must be contiguous within the record.	DDS
Key fields must be alphanumeric. They cannot be numeric only.	DDS
The value of the key used for sequencing must include all 8 bits of every byte.	DDS
A starting position for retrieving records cannot be specified.	The OVRDBF CL command
Select/omit level keywords cannot be used for the file.	DDS

When a record is copied using the COPY statement, DDS or DD format, a comment table appears in the COBOL source program listing identifying the characteristics controlled by DDS. This table includes:

- Key field number
- Key field name
- Keywords specified
- Data type.

An indexed file is identified by the ORGANIZATION IS INDEXED clause of the SELECT statement.

The key fields identify the records in an indexed file. The user specifies the key field in the RECORD KEY clause of the SELECT statement. The RECORD KEY data item must be defined within a record description for the indexed file. If there are multiple record descriptions for the file, only one need contain the RECORD KEY data-name. However, the same positions within the record description that contain the RECORD KEY data item are accessed in the other record descriptions as the KEY value for any references to the other record descriptions for that file.

An indexed file can be accessed sequentially, randomly by key, or dynamically.

Valid RECORD KEYS: The DDS for the file specifies the field(s) to be used as the key field. If the file has multiple key fields, the key fields must be contiguous in each record unless RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified.

When the DDS specifies only one key field for the file, the RECORD KEY must be a single field of the same length as the key field defined in the DDS.

If a COPY statement, DDS or DD format, is specified for the file, the RECORD KEY clause must specify one of the following:

- The name used in the DDS for the key field if the name is not a COBOL reserved word.
- The name used in the DDS for the key field with -DDS added to the end if the name is a COBOL reserved word.
- The data-name defined with the proper length and at the proper location in a program described record description for the file.
- EXTERNALLY-DESCRIBED-KEY. This keyword specifies that the key(s) defined in DDS for each record format are to be used for accessing the file. These keys can be noncontiguous. They can be defined at different positions within the record format.

When the DDS specifies multiple contiguous key fields, the RECORD KEY data-name must be a single field with its length equal to the sum of the lengths of the multiple key fields in the DDS. If a COPY statement, DDS or DD format, is specified for the file, there must also be a program described record description for the file that defines the RECORD KEY data-name with the proper length and at the proper position in the record.

Referring to a Partial Key

A generic START statement allows the use of a partial key. The KEY IS phrase is required.

“START Statement” in Chapter 5 lists the rules for specifying a search argument that refers to a partial key.

Figure 9-10 shows an example of generic STARTS using a program described file.

Figure 9-11 shows an example of generic STARTS using an externally described file.

```
                                COBOL SOURCE LISTING
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S COPYNAME  CHG/DATE
 7 000700 FILE-CONTROL.
 8 000800   SELECT FILE-1 ASSIGN TO DISK-FILE1
 9 000900   ACCESS IS DYNAMIC RECORD KEY IS FULL-NAME IN FILE-1
10 001000   ORGANIZATION IS INDEXED.
11 001100 DATA DIVISION.
12 001200 FILE SECTION.
13 001300 FD  FILE-1 LABEL RECORDS ARE STANDARD.
14 001400 01  RECORD-DESCRIPTION.
15 001500   03 FULL-NAME.
16 001600     05 LAST-AND-FIRST-NAMES.
17 001700       07 LAST-NAME                PIC X(20).
18 001800       07 FIRST-NAME                PIC X(20).
19 001900     05 MIDDLE-NAME                PIC X(20).
20 002000   03 LAST-FIRST-MIDDLE-INITIAL-NAME REDEFINES FULL-NAME
21 002100     PIC X(41).
22 002200   03 REST-OF-RECORD                PIC X(40).
002300/
23 002400 PROCEDURE DIVISION.
002500 START-PROGRAM.
24 002600   OPEN INPUT FILE-1.
002700*
002800* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
002900* "SMITH"
25 003000     MOVE "SMITH" TO LAST-NAME.
26 003100     START FILE-1 KEY IS EQUAL TO LAST-NAME
27 003200       INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR " LAST-NAME
28 003300         GO TO ERROR-ROUTINE.
003400*       .
003500*       .
003600*       .
003700*
003800* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
003900* "SMITH" AND A FIRST NAME OF ROBERT
29 004000     MOVE "SMITH" TO LAST-NAME.
30 004100     MOVE "ROBERT" TO FIRST-NAME.
31 004200     START FILE-1 KEY IS EQUAL TO LAST-AND-FIRST-NAMES
32 004300       INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
004400         LAST-AND-FIRST-NAMES
33 004500         GO TO ERROR-ROUTINE.
004600*       .
004700*       .
004800*       .
004900*
005000* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
005100* "SMITH", A FIRST NAME OF ROBERT, AND A MIDDLE INITIAL OF "M"
34 005200     MOVE "SMITH" TO LAST-NAME.
35 005300     MOVE "ROBERT" TO FIRST-NAME.
36 005400     MOVE "M" TO MIDDLE-NAME.
37 005500     START FILE-1 KEY IS EQUAL TO LAST-FIRST-MIDDLE-INITIAL-NAME
38 005600       INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
005700         LAST-FIRST-MIDDLE-INITIAL-NAME
39 005800         GO TO ERROR-ROUTINE.
005900
006000
006100 ERROR-ROUTINE.
40 006200   STOP RUN.
```

Figure 9-10. Generic STARTS Using a Program Described File

SEQNBR	*...	...	1	...	2	...	3	...	4	...	5	...	6	...	7	...	8	DATE	
100	A																		UNIQUE
200	A				R	RDE													TEXT(*RECORD DESCRIPTION*)
300	A					FNAME			20										TEXT(*FIRST NAME*)
400	A					MNAME			1										TEXT(*MIDDLE INITIAL NAME*)
500	A					MNAME			19										TEXT(*REST OF MIDDLE NAME*)
600	A					LNAME			20										TEXT(*LAST NAME*)
700	A					PHONE			10	0									TEXT(*PHONE NUMBER*)
800	A					DATA			40										TEXT(*REST OF DATA*)
900	A				K	LNAME													
1000	A				K	FNAME													
1100	A				K	MNAME													
1200	A				K	MNAME													

Figure 9-11 (Part 1 of 2). Generic STARTS Using an Externally Described File

```

                                COBOL SOURCE LISTING
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S COPYNAME  CHG/DATE
 7 000700 FILE-CONTROL.
 8 000800 SELECT FILE-1 ASSIGN TO DATABASE-NAMES
 9 000900 ACCESS IS DYNAMIC RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
10 001000 ORGANIZATION IS INDEXED.
11 001100 DATA DIVISION.
12 001200 FILE SECTION.
13 001300 FD FILE-1 LABEL RECORDS ARE STANDARD.
14 001400 01 RECORD-DESCRIPTION.
15 001500 COPY DDS-RDE IN NAMES-PUBS.
17 +000001 RDE
+000002* FROM FILE NAMES OF LIBRARY PUBS RDE
+000003* RECORD DESCRIPTION RDE
18 +000004 05 RDE. RDE
+000005* RECORD KEY FOR INDEXED FILE, KEY '0002 KEY FIELD NAME FNAME . RDE
19 +000006 06 FNAME PIC X(20). RDE
+000007* FIRST NAME RDE
+000008* RECORD KEY FOR INDEXED FILE, KEY 0003 KEY FIELD NAME MINAME . RDE
20 +000009 06 MINAME PIC X(1). RDE
+000010* MIDDLE INITIAL NAME RDE
+000011* RECORD KEY FOR INDEXED FILE, KEY 0004 KEY FIELD NAME MNAME . RDE
21 +000012 06 MNAME PIC X(19). RDE
+000013* REST OF MIDDLE NAME RDE
+000014* RECORD KEY FOR INDEXED FILE, KEY 0001 KEY FIELD NAME LNAME . RDE
22 +000015 06 LNAME PIC X(20). RDE
+000016* LAST NAME RDE
23 +000017 06 PHONE PIC S9(10) COMP-3. RDE
+000018* PHONE NUMBER RDE
24 +000019 06 DATA-DDS PIC X(40). RDE
+000020* REST OF DATA RDE
25 001600 66 MIDDLE-NAME RENAMES MINAME THRU MNAME.
001700/
26 001800 PROCEDURE DIVISION.
001900 START-PROGRAM.
27 002000 OPEN INPUT FILE-1.
002100*
002200* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME
002300* OF "SMITH"
28 002400 MOVE "SMITH" TO LNAME.
29 002500 START FILE-1 KEY IS EQUAL TO LNAME
30 002600 INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR " LNAME
31 002700 GO TO ERROR-ROUTINE.
002800* .
002900* .
003000* .
003100*
003200* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME
003300* OF "SMITH" AND A FIRST NAME OF ROBERT
32 003400 MOVE "SMITH" TO LNAME.
33 003500 MOVE "ROBERT" TO FNAME.
34 003600 START FILE-1 KEY IS EQUAL TO LNAME, FNAME
35 003700 INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
003800 LNAME " " FNAME
36 003900 GO TO ERROR-ROUTINE.
004000* .
004100* .
004200* .
004300*
004400* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
004500* "SMITH", A FIRST NAME OF ROBERT, AND A MIDDLE INITIAL OF "M"
37 004600 MOVE "SMITH" TO LNAME.
38 004700 MOVE "ROBERT" TO FNAME.
39 004800 MOVE "M" TO MINAME.
40 004900 START FILE-1 KEY IS EQUAL TO LNAME, FNAME, MINAME
41 005000 INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
005100 LNAME SPACE FNAME SPACE MINAME
42 005200 GO TO ERROR-ROUTINE.
005300
005400
005500 ERROR-ROUTINE.
43 005600 STOP RUN.

```

Figure 9-11 (Part 2 of 2). Generic STARTS Using an Externally Described File

Logical File Considerations

When a logical file with multiple record formats, each having associated key fields, is processed as an indexed file in COBOL, the following restrictions and considerations apply:

- The FORMAT phrase must be specified on all WRITE statements to the file.
- If the access mode is RANDOM or DYNAMIC, and the DUPLICATES phrase is not specified for the file, the FORMAT phrase must be specified on all DELETE and REWRITE statements.
- When the FORMAT phrase is not specified, only the portion of the RECORD KEY data item that is common to all record formats for the file is used by the system as the key for the I-O statement. When the FORMAT phrase is specified, only the portion of the RECORD KEY data item that is defined for the specified record format is used by the system as the key. See the *CPF Programmer's Guide* for more information on logical file processing.
- When *NONE is specified as the first key field for any format in a file, records can only be accessed sequentially. When a file is read randomly:
 - If a format name is specified, the first record with the specified format is returned.
 - If a format name is not specified, the first record in the file is returned.

In both cases, the value of the RECORD KEY data item is ignored.

- For a program defined key field:
 - Key fields within each record format must be contiguous.
 - The first key field for each record format must begin at the same relative position within each record.
 - The length of the RECORD KEY data item must be equal to the length of the longest key for any format in the file.
- For an EXTERNALLY-DESCRIBED-KEY:
 - Key fields within each record format can be noncontiguous.
 - The key fields can begin at different positions in each record format.

Figures 9-12 and 9-13 show examples of how to use DDS to describe the access path for indexed files.

File	Keying Instruction	Graphic								Description	Page	of
Programmer	Date	Key										

Sequence Number	Form Type	And/Or/Comment (A/O/?)	Conditioning					Name	Length	Reference (R)	Data Type (B/A/P/S/B/A/S/X/Y/N/I/M)	Decimal Positions	Usage (B/O)/E/N/M)	Location		Functions
			Condition Name											Line	Pos	
			Indicator	Not (N)	Indicator	Not (N)	Indicator									
1																
2																
3																
4																
5																
6	A						R	FORMATA								PFILE(ORDDTLP)
7	A															Text('Access Path for Indexed File')
8								FLDA			14					
9								ORDER			5	0				
10								FLDB			101					
11																
12																
13																
14																
15																
16																
17																
18																
19																
20																
21																
22																
23																
24																
25																
26																
27																
28																
29																
30																
31																
32																
33																
34																
35																
36																
37																
38																
39																
40																
41																
42																
43																
44																
45																
46																
47																
48																
49																
50																
51																
52																
53																
54																
55																
56																
57																
58																
59																
60																
61																
62																
63																
64																
65																
66																
67																
68																
69																
70																
71																
72																
73																
74																
75																
76																
77																
78																
79																
80																

*Number of sheets per pad may vary slightly.

Data description specifications must be used to create the access path for a program described indexed file.

In the DDS for the record format FORMATA for the logical file ORDDTLL, the field ORDER, which is 5 digits long, is defined as the key field, and is in packed format. The definition of ORDER as the key field establishes the keyed access for this file. Two other fields, FLDA and FLDB, describe the remaining positions in this record as character fields.

The program described input field ORDDTLL is described in the FILE-CONTROL section in the SELECT clause as an indexed file.

The COBOL descriptions of each field in the FD entry must agree with the corresponding description in the DDS file. The RECORD KEY data item must be defined as a 5-digit numeric integer beginning in position 15 of the record.

Figure 9-12. Using Data Description Specifications to Define the Access Path for an Indexed File

File	Keying Instruction	Graphic							
Programmer		Date	Key						

Description	Page	of
-------------	------	----

A Sequence Number	Form Type 6 And/Or Comment (A/O/*) 7 Not (N)	Conditioning				Name	Length	Reference (R)	Data Type (B A/P/S/B A/S/X/N/I/M) 35 36 37 38 39 40 41 42 43 44	Location		Functions
		Condition Name								Line	Pos	
		Indicator	Not (N)	Indicator	Not (N)							
1	A					FORMAT						PFILE(ORDDTLP) TEXT('Access Path for Indexed File')
	A					FLDA	14					
	A					ORDER	5					
	A					ITEM	5					
	A					FLOB	96					
	A					K ORDER						
	A					K ITEM						

*Number of sheets per pad may vary slightly.

In this example, the data description specifications define two key fields for the record format FORMAT in the logical file ORDDTLL. For the two fields to be used as a composite key for a program described indexed file, the key fields must be contiguous in the record.

The COBOL description of each field must agree with the corresponding description in the DDS file. A 10-character item beginning in position 15 of the record must be defined in the RECORD KEY clause of the file-control entry. The COBOL descriptions of the DDS fields ORDER and ITEM would be subordinate to the 10-character item defined in the RECORD KEY clause.

Figure 9-13. Using Data Description Specifications to Define the Access Path (a Composite Key) for an Indexed File

COBOL Relative File: A COBOL relative file is a file to be processed by a relative record number. To process a file by relative record number, ORGANIZATION IS RELATIVE must be specified in the SELECT statement for the file. A relative file can be accessed sequentially, randomly by record number, or dynamically.

To write standard ANS COBOL X3.23-1974 to access a relative file, the file must be created with certain characteristics. The following table lists these characteristics and what controls them.

Characteristic	Control
The file must be a physical file.	The CRTPF CL command
The file cannot be a shared file.	The CRTPF CL command
No key can be specified for the file.	DDS
A starting position for retrieving records cannot be specified.	The OVRDBF CL command
Select/omit level keywords cannot be used for the file.	DDS

For a COBOL file with an organization of `RELATIVE`, the Reorganize Physical File Member (RGZPFM) CL command can:

- Remove all deleted records from the file. Since COBOL initializes all relative file records to deleted records, any record that has not been explicitly written will be removed from the file. This causes the relative record numbers of all records after the first deleted record in the file to change.
- Change the relative record numbers if the file has a key and the arrival sequence is changed to match a key sequence (with the `KEYFILE` parameter).

Either result of the RGZPFM CL command causes the COBOL concept of a relative file to change.

COBOL Sequential File: A COBOL sequential file is a file in which records are processed in the order in which they were placed in the file; that is, in arrival sequence. For example, the tenth record placed in the file occupies the tenth record position and is the tenth record to be processed. To process a file as a sequential file, `ORGANIZATION IS SEQUENTIAL` must be specified in the `SELECT` clause, or the `ORGANIZATION` clause can be omitted. A sequential file can only be accessed sequentially.

To write standard ANS COBOL X3.23-1974 to access a sequential file, the file must be created with certain characteristics. The following table lists these characteristics and what controls them.

Characteristic	Control
The file must be a physical file.	The CRTPF CL command
The file cannot be a shared file.	The CRTPF CL command
The device specified in the assignment-name must match the actual device to which the file is assigned.	
No key can be specified for the file.	DDS
The file must have a file-type of data.	The CRTPF command
Field editing cannot be used.	DDS
Line and position information cannot be specified.	DDS
Spacing and shipping keywords cannot be specified.	DDS

Characteristic	Control
Indicators cannot be used.	DDS
System-supplied functions such as date, time, and page number cannot be used.	DDS
Select/omit level keywords cannot be used for the file.	DDS

COBOL File Organization and System/38 File Access Path

Considerations: A file with a keyed sequence access path can be processed in COBOL as a file with INDEXED, RELATIVE, or SEQUENTIAL organization.

To process a keyed sequence file as a relative file in COBOL, the file must be a physical file, or a logical file whose members are based on one physical file member. To process a keyed sequence file as a sequential file in COBOL, the file must be a physical file, or a logical file that is based on one physical file member and that does not contain select/omit logic.

A file with an arrival sequence access path can be processed in COBOL as a file with RELATIVE or SEQUENTIAL organization. However, the file must be a physical file or a logical file where each member of the logical file is based on only one physical file member.

When sequential access is specified for a logical file, records in the file are accessed through the access path created by the user with create file options.

File Processing Methods

Figure 9-14 shows the valid processing methods and expected operation for combinations of organization, access mode, open state, I-O verb, and I-O verb modifiers.

Programming Note: All physical data base files that are opened for output are cleared. Data base files with RELATIVE organization are also initialized with deleted records, which is necessary for successful relative file processing on System/38. Relative files should be cleared and initialized with deleted records before they are used when the first OPEN statement for the file is not OPEN OUTPUT. See the discussion of the CLRPFM and INZPFM commands in the *CL Reference Manual*. The RECORDS parameter in the INZPFM command must specify *DLT. Overrides are applied when COBOL performs the clear and initialize operations, but overrides are not applied when the user performs the clear and initialize operations with CL commands.

Users can expect lengthy delays in OPEN OUTPUT processing for extremely large relative files (over 1,000,000 records), such as when *NOMAX is specified on the create file command.

Device files, such as QTAPE and QDKT, must be initialized by the user. Refer to the *CL Reference Manual* for information on the INZTAP and INZDKT commands.

Descending File Considerations

Files created with a descending keyed sequence (in DDS) cause the READ statement NEXT, PRIOR, FIRST, and LAST phrases to work in a fashion exactly opposite that of a file with an ascending key sequence. For example, READ FIRST retrieves the record with the highest key value, and READ LAST retrieves the record with the lowest key value. Files with a descending key sequence also cause the START qualifiers to work in the opposite manner. For example, START GREATER THAN positions the current record pointer to a record with a key less than the current key.

ORG	ACC	DEV	OPEN	READ	WRITE	START	REWRITE	DELETE	CLOSE	FORMAT	SELECT CLAUSE KEY IS
S	S	ANY	INPUT	X					X		
S	S	ANY	OUTPUT		X(F1)				X	A1	
S	S	ANY	I-O	X			X		X		
S	S	ANY	EXTEND		X				X		
I	S	D/DB	INPUT	X		X			X	B1	C1
I	S	D/DB	OUTPUT		X(F1)				X	B1	C1
I	S	D/DB	I-O	X		X	X	X	X	B1	C1
I	R	D/DB	INPUT	X					X	B1	D1
I	R	D/DB	OUTPUT		X(F1)				X	B1	D1
I	R	D/DB	I-O	X	X		X	X	X	B1	D1
I	D	D/DB	INPUT	X		X			X	B1	D1
I	D	D/DB	OUTPUT		X(F1)				X	B1	D1
I	D	D/DB	I-O	X	X	X	X	X	X	B1	D1
R	S	D/DB	INPUT	X		X			X		C1
R	S	D/DB	OUTPUT		X(G1)				X		C1
R	S	D/DB	I-O	X		X	X	X	X		C1
R	R	D/DB	INPUT	X					X		E1
R	R	D/DB	OUTPUT		X(G1)				X		E1
R	R	D/DB	I-O	X	X		X	X	X		E1
R	D	D/DB	INPUT	X		X			X		E1
R	D	D/DB	OUTPUT		X(G1)				X		E1
R	D	D/DB	I-O	X	X	X	X	X	X		E1
T	S	W	I-O	X	X				X	H1	
T	D	W	I-O	X(K1)	X(K1)		X		X	I1	J1
ORG: S = Sequential R = Relative I = Indexed T = TRANSACTION				ACC: S = Sequential R = Random D = Dynamic			DEV: ANY = Any device D = DISK DB = DATABASE W = WORKSTATION				

Figure 9-14. Processing Methods Summary Chart

The following paragraphs explain the keys used in Figure 9-14.

- X The combination is allowed.
- A1 The FORMAT phrase is required for FORMATFILE files with multiple formats, and is not allowed for all other device files.
- B1 The FORMAT phrase is optional for DATABASE files, and not allowed for DISK files. If the FORMAT phrase is not specified, the default format name of the file is used. The default format name of the file is the first format name defined in the file.

The special register, DB-FORMAT-NAME, can be used to retrieve the format name used on the last successful I-O operation.

- C1 The SELECT clause KEY phrase is ignored except for the START statement. If the KEY phrase is not specified on the START statement, the RECORD KEY phrase or the RELATIVE KEY phrase in the SELECT clause is used and KEY = is assumed.
- D1 The SELECT clause KEY phrase is used except for the START statement. If the KEY phrase is not specified on the START statement, the RECORD KEY phrase in the SELECT clause is used and KEY = is assumed.

NEXT, PRIOR, FIRST, or LAST can be specified only for the READ statement for DATABASE files with DYNAMIC access. If NEXT, PRIOR, FIRST, or LAST is specified, the SELECT clause KEY phrase is ignored.

- E1 The SELECT clause RELATIVE KEY phrase is used.

The NEXT phrase can be specified only for the READ statement for a file with DYNAMIC access mode. If NEXT is specified, the SELECT clause KEY phrase is ignored.

The RELATIVE KEY data item is updated with the relative record number for files with sequential access on READ operations.

- F1 A physical file opened for output is cleared.
- G1 A physical file opened for output is cleared and initialized to deleted records.
- H1 The FORMAT phrase is required for the WRITE statement.
- I1 The FORMAT phrase is required to distinguish between the subfile records and the subfile control record. The WRITE FORMAT IS control-record-format-name displays the subfile, but a READ FORMAT IS control-record-format-name is required to allow data to be entered and to cause the operator input for the subfile records on the display to be placed in the subfile.
- J1 The SELECT clause RELATIVE KEY phrase is used for READ, WRITE, and REWRITE statements that use the SUBFILE phrase, except that the READ SUBFILE NEXT MODIFIED uses the current system relative record number rather than the RELATIVE KEY data item. The RELATIVE KEY data item is

updated with the relative record number for subfile records for READ statements with the NEXT MODIFIED clause.

- K1 The SUBFILE phrase is required when an I-O operation deals with a particular record rather than an entire file.

Commitment Control Considerations

Commitment control is a function that allows:

- Synchronization of changes to data base files within the same job
- Cancellation of changes that should not be permanently entered into the database
- Locking of records being changed until changes are complete
- Techniques for recovering from job or system failure.

In some applications, it is desirable to synchronize changes to data base records. If the program determines the changes are valid, the changes are then permanently made to the data base (a COMMIT statement is executed). If the changes are not valid, or if a problem occurs during processing, the changes can be canceled (a ROLLBACK statement is executed). (When a file is cleared after being opened for OUTPUT, execution of a ROLLBACK does not restore cleared records to the file.) Changes made to records in a file that is *not* under commitment control are always permanent. Such changes are never affected by subsequent COMMIT or ROLLBACK statements.

Each point where a COMMIT or ROLLBACK is successfully executed is a commitment boundary. (If no COMMIT or ROLLBACK has yet been issued in a program, a commitment boundary is created by the first open of any file under commitment control.) The committing or rolling back of changes only affects changes made since the previous commitment boundary.

The synchronizing of changes at commitment boundaries makes restart or recovery procedures after a failure easier. For more information see “Recovery After Failure” later in this chapter.

When commitment control is used for data base files, records in those files are subject to either a high lock level LCKLVL (*ALL) or a low lock level LCKLVL (*CHG). With a low lock level (*CHG), all records that are changed (rewritten, deleted, or added) in files under commitment control are locked until a COMMIT or ROLLBACK statement is successfully executed. With a high lock level (*ALL), *all* records accessed, whether for input or output, are locked until a COMMIT or ROLLBACK is successfully executed. For both record locking levels, no other job can modify data in locked records until the COMMIT or ROLLBACK has been successfully completed. (A locked record can only be modified within the same job and through the same physical or logical file.)

The lock level also governs whether locked records can be read. With a high lock level (*ALL), you cannot read locked records in a data base file.

With a low lock level (*CHG), you can read locked records in a data base file, provided the file is opened as INPUT in your job.

Other jobs, where files are *not* under commitment control, can always read locked records, regardless of the lock level used, provided the files are opened as INPUT. Because it is possible in some cases for other jobs to read locked records, data can be accessed *before it is permanently committed to a data base*. If a ROLLBACK statement is executed *after* another job has read locked records, the data accessed will not reflect the contents of the data base.

Figure 9-15 shows record locking considerations for files with and without commitment control.

VERB	OPEN MODE	LOCK LEVEL		DURATION OF RECORD LOCK	
				Next I-O Operation	COMMIT or ROLLBACK
DELETE	I-O	Without Commitment Control		DELETE	
		With Commitment Control	*CHG	○	_____
			*ALL	○	_____
READ	INPUT	Without Commitment Control		READ	
		With Commitment Control	*CHG	○	_____
			*ALL	○	_____
READ	I-O	Without Commitment Control		READ	
		With Commitment Control	*CHG	○	_____
			*ALL	○	_____
REWRITE	I-O	Without Commitment Control		REWRITE	
		With Commitment Control	*CHG	○	_____
			*ALL	○	_____
START	INPUT	Without Commitment Control		START	
		With Commitment Control	*CHG	○	_____
			*ALL	○	_____
START	I-O	Without Commitment Control		START	
		With Commitment Control	*CHG	○	_____
			*ALL	○	_____
WRITE	I-O	Without Commitment Control		WRITE	
		With Commitment Control	*CHG	○	_____
			*ALL	○	_____
WRITE	OUTPUT	Without Commitment Control		WRITE	
		With Commitment Control	*CHG	○	_____
			*ALL	○	_____

Figure 9-15. Record Locking Considerations with and without Commitment Control

A file under commitment control can be closed or opened without affecting the status of changes made since the last commitment boundary. A COMMIT must still be issued to make the changes permanent, or a ROLLBACK issued to cancel the changes. A COMMIT statement, when executed, leaves files in the same open or closed state as before execution.

All files under commitment control within the same job must be journaled to the same journal. For more information about journal management and its related functions, and for more information about commitment control, refer to the *CPF Programmer's Guide*.

Commitment control must also be specified outside the COBOL language through the CPF control language (CL). The Begin Commitment Control (BGNCMTCTL) CL command establishes the capability for commitment control and sets the level of record locking at the high level (*ALL), or the low level (*CHG). The BGNCMTCTL command does not automatically initiate commitment control for a file. That file must also be specified in the COMMITMENT CONTROL clause of the I-O-CONTROL paragraph within the COBOL program. The commitment control environment is normally terminated by using the End Commitment Control (ENDCMTCTL) CL command. This causes any uncommitted changes for data base files under commitment control to be cancelled. (An implicit ROLLBACK is executed.) Refer to the *CL Reference Manual* for more information on the BGNCMTCTL and ENDCMTCTL commands.

For more information about commitment control see the *CPF Programmer's Guide*.

Programming Note: The ability to prevent reading of uncommitted data that has been changed is a function of commitment control and is only available if you are running under commitment control. Normal (non-commit) data base support is not changed by the commitment control extension, and allows reading of locked records when a file that is opened only for input is read. Try to use files consistently. Typically, files should always be run under commitment control or never be run under commitment control.

Figure 9-16 shows an example of the use of commitment control.

The following example illustrates a possible use of commitment control in a banking environment. The program processes transactions for transferring funds from one account to another. If no problems occur during the transaction, the changes are committed to the data base file. If the transfer is invalid due to improper account number or insufficient funds, a ROLLBACK is issued to cancel the changes.

STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN 5 COPYNAME

```

1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID. ACCOUNT.
3 000300 AUTHOR. IBM CANADA LABORATORY.
4 000400 INSTALLATION. TORONTO LABORATORY.
5 000500 ENVIRONMENT DIVISION.
6 000600 CONFIGURATION SECTION.
7 000700 SOURCE-COMPUTER. IBM-S38.
8 000800 OBJECT-COMPUTER. IBM-S38.
9 000900 INPUT-OUTPUT SECTION.
10 001000 FILE-CONTROL.
11 001100 SELECT ACCOUNT-FILE
12 001200 ASSIGN TO DATABASE-ACCTMST
13 001300 ORGANIZATION IS INDEXED
14 001400 ACCESS IS DYNAMIC
15 001500 RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
16 001600 FILE STATUS IS ACCOUNT-FILE-STATUS.
17 001700 SELECT DISPLAY-FILE
18 001800 ASSIGN TO WORKSTATION-ACCTFMST-SI 1
19 001900 ORGANIZATION IS TRANSACTION.
002000*****
002100*
20 002200 I-O-CONTROL. COMMITMENT CONTROL FOR ACCOUNT-FILE.
002300*
002400*****
21 002500 DATA DIVISION.
22 002600 FILE SECTION.
23 002700 FD ACCOUNT-FILE
24 002800 LABEL RECORDS ARE STANDARD.
25 002900 01 ACCOUNT-RECORD.
26 003000 COPY DDS-ALL-FORMATS OF ACCTMST.
28 +000001 05 ACCTHST-RECORD PIC X(82).
29 +000002
+000003* I-O FORMAT:ACCTREC FROM FILE ACCTMST OF LIBRARY COMMIT
+000004*
+000005*THE KEY DEFINITIONS FOR RECORD FORMAT ACCTREC
+000006* NUMBER NAME RETRIEVAL TYPE ALTSEQ
+000007* 0001 ACCNTKEY REDEFINES ACCTMST-RECORD.
30 +000008 05 ACCTREC REDEFINES ACCTMST-RECORD.
31 +000009 06 ACCTKEY PIC S9(5).
32 +000010 06 NAME PIC X(20).
33 +000011 06 ADDR PIC X(20).
34 +000012 06 CITY PIC X(20).
35 +000013 06 STATE PIC X(2).
36 +000014 06 ZIP PIC S9(5).
37 +000015 06 BALANCE PIC S9(18)V(2).
38 003100 FD DISPLAY-FILE
39 003200 LABEL RECORDS ARE OMITTED.
40 003300 01 DISPLAY-REC.
41 003400 COPY DDS-ALL-FORMATS OF ACCTFMST.
43 +000001 05 ACCTFMST-RECORD PIC X(24).
44 +000002
+000003* INPUT FORMAT:ACCTPMT FROM FILE ACCTFMST OF LIBRARY COMMIT
+000004* CUSTOMER ACCOUNT PROMPT
45 +000005 05 ACCTPMT-I REDEFINES ACCTFMST-RECORD.
46 +000006 06 ACCTFROM PIC S9(5).

```

Figure 9-16 (Part 2 of 6). Example of Use of Commitment Control

```

5714CR1 R05 MO0 830A10          COBOL SOURCE LISTING          ACCOUNT          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S CJPYNAME

47 +000007      06 ACCTTD          PIC S9(5).                <-ALL-FMTS
48 +000008      06 TRANSAMT       PIC S9(8)V9(2).          <-ALL-FMTS
49 +000009
+000010* OUTPUT FORMAT:ACCTPMT   FROM FILE ACCTFMTS OF LIBRARY COMMIT
+000011*      CUSTOMER ACCOUNT PROMPT
+000012*      05 ACCTPMT-D       REDEFINES ACCTFMTS-RECORD. <-ALL-FMTS
50 +000013      INPUT FORMAT:ERRFMT FROM FILE ACCTFMTS OF LIBRARY COMMIT
+000014*      05 ERRFMT-I       REDEFINES ACCTFMTS-RECORD. <-ALL-FMTS
51 +000017      OUTPUT FORMAT:ERRFMT FROM FILE ACCTFMTS OF LIBRARY COMMIT
+000018*      05 ERRFMT-D       REDEFINES ACCTFMTS-RECORD. <-ALL-FMTS
+000019*
+000020*      05 ERRFMT-D       REDEFINES ACCTFMTS-RECORD. <-ALL-FMTS
52 003500 WORKING-STORAGE SECTION.
53 003600 77 ACCOUNT-FILE-STATUS PIC XX.
54 003700 77 IND-ON   PIC I VALUE B"1".
55 003800 77 IND-OFF  PIC I VALUE B"0".
56 003900 61 FROM-ACCOUNT-RECORD.
57 004000 COPY DDS-ALL-FORMATS OF ACCTMST.
59 +000001      05 ACCTMST-RECORD PIC X(32).                <-ALL-FMTS
60 +000002
+000003* I-D FORMAT:ACCTMSTREC   FROM FILE ACCTMST   OF LIBRARY COMMIT
+000004*
61 +000005      05 ACCTMSTREC   REDEFINES ACCTMST-RECORD. <-ALL-FMTS
62 +000006      06 ACCTKEY     PIC S9(5).                <-ALL-FMTS
63 +000007      06 NAME       PIC X(20).                <-ALL-FMTS
64 +000008      06 ADDR       PIC X(20).                <-ALL-FMTS
65 +000009      06 CITY       PIC X(20).                <-ALL-FMTS
66 +000010      06 STATE      PIC X(2).                 <-ALL-FMTS
67 +000011      06 ZIP        PIC S9(5).                <-ALL-FMTS
68 +000012      06 BALANCE    PIC S9(8)V9(2).          <-ALL-FMTS
69 004100 01 TN-ACCOUNT-RECORD.
70 004200 COPY DDS-ALL-FORMATS OF ACCTMST.
72 +000001      05 ACCTMST-RECORD PIC X(32).                <-ALL-FMTS
73 +000002
+000003* I-D FORMAT:ACCTMSTREC   FROM FILE ACCTMST   OF LIBRARY COMMIT
+000004*
74 +000005      05 ACCTMSTREC   REDEFINES ACCTMST-RECORD. <-ALL-FMTS
75 +000006      06 ACCTKEY     PIC S9(5).                <-ALL-FMTS
76 +000007      06 NAME       PIC X(20).                <-ALL-FMTS
77 +000008      06 ADDR       PIC X(20).                <-ALL-FMTS
78 +000009      06 CITY       PIC X(20).                <-ALL-FMTS
79 +000010      06 STATE      PIC X(2).                 <-ALL-FMTS
80 +000011      06 ZIP        PIC S9(5).                <-ALL-FMTS
81 +000012      06 BALANCE    PIC S9(8)V9(2).          <-ALL-FMTS
82 004300 01 DISPFIL-INDICS.
83 004400 COPY DDS-ALL-FORMATS-INDIC OF ACCTFMTS. 3
85 +000001      05 ACCTFMTS-RECORD.                <-ALL-FMTS
86 +000002
+000003* INPUT FORMAT:ACCTPMT   FROM FILE ACCTFMTS OF LIBRARY COMMIT
+000004*      CUSTOMER ACCOUNT PROMPT
87 +000005      05 ACCTPMT-I-INDIC.
88 +000006      07 IN15        PIC I INDIC 15.
+000007*      END OF PROGRAM                                <-ALL-FMTS

```

Figure 9-16 (Part 3 of 6). Example of Use of Commitment Control


```

5714CB1 R05 M00 830610          COBOL SOURCE LISTING          ACCOUNT          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S  CDPYNAME

 89 *000008          07 IN97          PIC 1 INDIC 97.          <-ALL-FMTS
*000009*          INVALID TO ACCOUNT NUMBFR          <-ALL-FMTS
 90 *000010          07 IN98          PIC 1 INDIC 98.          <-ALL-FMTS
*000011*          INSUFFICIENT FUNDS IN FROM ACCOUNT          <-ALL-FMTS
 91 *000012          07 IN99          PIC 1 INDIC 99.          <-ALL-FMTS
*000013*          INVALID FROM ACCOUNT NUMBER          <-ALL-FMTS
 92 *000014          06 ACCTPMT-0-INDIC.          <-ALL-FMTS
*000015* OUTPUT FORMAT:ACCTPMT FROM FILE ACCTFMTS OF LIBRARY COMMIT
*000016* CUSTOMER ACCOUNT PROMPT          <-ALL-FMTS
 93 *000017          07 IN97          PIC 1 INDIC 97.          <-ALL-FMTS
 94 *000018          07 IN97          INVALID TO ACCOUNT NUMBER          <-ALL-FMTS
*000019*          PIC 1 INDIC 98.          <-ALL-FMTS
 95 *000020          07 IN98          INSUFFICIENT FUNDS IN FROM ACCOUNT          <-ALL-FMTS
*000021*          PIC 1 INDIC 99.          <-ALL-FMTS
 96 *000022          07 IN99          INVALID FROM ACCOUNT NUMBER          <-ALL-FMTS
*000023*          <-ALL-FMTS
 97 *000024          06 ERRFMT-0-INDIC.          <-ALL-FMTS
*000025* INPUT FORMAT:ERRFMT FROM FILE ACCTFMTS OF LIBRARY COMMIT
*000026*          <-ALL-FMTS
*000027*          06 ERRFMT-1-INDIC.          <-ALL-FMTS
 98 *000028          07 IN95          PIC 1 INDIC 95.          <-ALL-FMTS
*000029* OUTPUT FORMAT:ERRFMT FROM FILE ACCTFMTS OF LIBRARY COMMIT
*000030*          <-ALL-FMTS
 99 *000031          06 ERRFMT-0-INDIC.          <-ALL-FMTS
*000032*          07 IN95          PIC 1 INDIC 95.          <-ALL-FMTS
100 *000033          07 IN96          PIC 1 INDIC 96.          <-ALL-FMTS
101 *000033          07 IN96          PIC 1 INDIC 96.          <-ALL-FMTS
102 004500 PROCEDURE DIVISION.
004600
004700 DECLARATIVES.
004800 ERROR-SECTION SECTION.
004900 USE AFTER STANDARD EXCEPTION PROCEDURE ON ACCOUNT-FILE.
005000 ERROR-PARAGRAPH.
103 005100 IF ACCOUNT-FILE-STATUS NOT = "23" THEN 4
104 005200 MOVE IND-ON TO IN96 OF ERRFMT-0-INDIC
005300 ELSE
105 005400 MOVE IND-ON TO IN95 OF ERRFMT-0-INDIC.
106 005500 WRITE DISPLAY-REC FORMAT IS "ERRFMT" 5
005600 INDICATORS ARE ERRFMT-0-INDIC.
107 005700 CLOSE DISPLAY-FILE.
005800 ACCOUNT-FILE.
108 005900 STOP RUN.
006000 END DECLARATIVES.
006100
006200 BEGIN.
006300
109 006400 OPEN I-O ACCOUNT-FILE.
006500 I-O DISPLAY-FILE.
110 006600 MOVE ZEROS TO ACCTPMT-1-INDIC.
006700 ACCTPMT-0-INDIC.
111 006800 PERFORM WRITE-READ-DISPLAY.
112 006900 PERFORM VERIFY-ACCOUNT-NO UNTIL IN15 = IND-ON.
113 007000 CLOSE DISPLAY-FILE.
007100 ACCOUNT-FILE.
114 007200 STOP RUN.
007300

```

Figure 9-16 (Part 4 of 6). Example of Use of Commitment Control

```

5714CB1 R05 MOD 833610          COBOL SOURCE LISTING          ACCOUNT          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN 5 COPYNAME

007400 VERIFY-ACCOUNT-NO.
007500
115 007600 PERFORM VERIFY-ACCOUNT-FROM.
116 007700 IF IN99 OF ACCTPMT-D-INDIC = IND-OFF THEN 6
117 007800 PERFORM VERIFY-ACCOUNT-TO.
118 007900 PERFORM WRITE-READ-DISPLAY.
008000
008100 VERIFY-ACCOUNT-FROM.
008200
119 008300 MOVE ACCTFROM TO ACNTKEY IN ACCOUNT-RECORD.
120 008400 READ ACCOUNT-FILE INTO FROM-ACCOUNT-RECORD
121 008500 INVALID KEY MOVE IND-ON TO IN99 OF ACCTPMT-D-INDIC.
008600
008700 VERIFY-ACCOUNT-TO.
008800
122 008900 MOVE ACCTTO TO ACNTKEY IN ACCOUNT-RECORD.
123 009000 READ ACCOUNT-FILE INTO TO-ACCOUNT-RECORD
124 009100 INVALID KEY MOVE IND-ON TO IN97 OF ACCTPMT-D-INDIC. 7
125 009200 IF IN97 OF ACCTPMT-D-INDIC = IND-ON THEN
009300*****
009400*
009500 ROLLBACK 8
126 009600*
009700*****
009800 ELSE
127 009900 PERFORM UPDATE-ACCOUNT.
010000
010100 UPDATE-ACCOUNT.
010200
128 010300 SUBTRACT TRANSMT FROM BALANCE IN FROM-ACCOUNT-RECORD.
129 010400 ADD TRANSMT TO BALANCE IN TO-ACCOUNT-RECORD.
130 010500 REWRITE ACCOUNT-RECORD FROM FROM-ACCOUNT-RECORD.
131 010600 REWRITE ACCOUNT-RECORD FROM TO-ACCOUNT-RECORD.
132 010700 IF BALANCE IN FROM-ACCOUNT-RECORD < 0 THEN
133 010800 MOVE IND-ON TO IN98 OF ACCTPMT-D-INDIC.
010900*****
011000*
011100 ROLLBACK 9
134 011200*
011300*****
011400 ELSE
011500*****
011600*
011700 COMMIT. 10
135 011800*
011900*****
012000
012100 WRITE-READ-DISPLAY.
012200
136 012300 WRITE DISPLAY-REC FORMAT IS "ACCTPMT"
012400 INDICATORS ARE ACCTPMT-D-INDIC. 11
137 012500 MOVE ZEROS TO ACCTPMT-I-INDIC.
012600 ACCTPMT-D-INDIC.
138 012700 READ DISPLAY-FILE RECORD
012800 INDICATORS ARE ACCTPMT-I-INDIC.

***** END OF SOURCE *****

```

Figure 9-16 (Part 5 of 6). Example of Use of Commitment Control

- 1** A separate indicator area is provided for the program.
- 2** The COMMITMENT CONTROL clause specifies files to be placed under commitment control. Any files named in this clause are affected by the COMMIT and ROLLBACK verbs.
- 3** The COPY statement, DDS format, with the indicator attribute INDIC, defines data description entries in WORKING-STORAGE for the indicators to be used in the program.
- 4** IN96 is set if there is an invalid file status.
- 5** IN95 is set if there is an INVALID KEY condition on the REWRITE operation.
- 6** IN99 is set if the entered account number is invalid for the account that money is being transferred to.
- 7** IN97 is set if the entered account number is invalid for the account that money is being transferred to.
- 8** If an INVALID KEY condition occurs on the READ, a ROLLBACK is used and the record lock placed on the record after the first READ is released.
- 9** If the transfer of funds is invalid (an indicator has been set), the ROLLBACK statement is executed. All changes made to data base files under commitment control are canceled.
- 10** If the transfer of funds was valid (no indicators have been set), the COMMIT statement is executed, and all changes made to data base files under commitment control then become permanent.
- 11** The INDICATORS phrase is required for indicators controlled by Command Function keys on the work station screen.

Figure 9-16 (Part 6 of 6). Example of Use of Commitment Control

Exceptions and Some of Their Causes

MCH1202 data exception:

- A numeric elementary item has been used as a source when no valid data has been previously stored in it. The item should have a VALUE clause, or a MOVE statement should be used to initialize its value.
- An attempt has been made to place nonnumeric data in a numeric item.
- Bad data was written to a subfile earlier in the program. The subfile data is not validated until it is written to the screen, so the 1202 error can occur on the WRITE of a subfile control record, but the bad data was actually put to the subfile earlier.

MCH3601 invalid pointer:

- The OPEN for the file was unsuccessful and the execution of any other I-O statement for that file is attempted. The file status should be checked before any other I-O is attempted.
- A reference is made to a record or a field within a record and the associated file has been closed or has never been opened.

MCH3402 reference destroyed object:

- A reference is made to a record or a field within a record and the associated data base file has been closed.

CPF2415 end of request:

- An attempt has been made to accept input from the job input stream while the system is running in batch mode and no input is available.

System/38 COBOL Programming Considerations

Performance Considerations

Segmentation

Use of segmentation increases the compile and execution times of the COBOL program. The segmentation feature is provided only for compatibility with other systems. It is not necessary to be concerned with storage management when using System/38 COBOL programs.

Debugging

COBOL source language debugging is provided to help the COBOL programmer debug a program that is not functioning as expected. Use of this facility increases the compile and execution times of a COBOL program.

Data Formats

Data described as packed (COMP or COMP-3) provides better performance than data described as binary (COMP-4) or zoned decimal (DISPLAY) at execution time.

*NORANGE Option

This option of the GENOPT parameter of the CRTCLPGM command removes the execution-time checks for subscript ranges. If frequent references to tables are made and the subscripts always reference elements within the table, use of this option can improve performance.

Indicators

If you use indicators in a separate indicator area (INDARA keyword specified in DDS) instead of in the record area, the use of the OCCURS clause to specify a table with up to 99 indicators can improve performance.

Commitment Control

Generally, the use of commitment control increases the execution time of a COBOL program. In addition, the record locking which results from the use of commitment control by a job may cause delays for other users attempting to access the same file.

Program Loops

When a program repeatedly executes the same series of instructions, and it is apparent that this will continue indefinitely, the program is in a loop. To identify loops, you can use information known about the program itself, as follows:

- Time: If the actual run time is substantially exceeding the expected run time, the program could be in a loop.
- I-O operations: If no input/output operations are taking place and I-O is expected to be occurring repeatedly, the program is probably in a loop.

Tracing a Loop in a Program

Frequently, a loop encompasses many instructions in a program. In this case, the user can use the COBOL debugging features as described in Chapter 6, or the System/38 debugging capabilities of Trace as described in Chapter 10.

Errors That Can Cause a Loop

A PERFORM statement with an UNTIL clause can cause a loop when the condition specified in the UNTIL clause cannot be met. For example:

```
PERFORM .. UNTIL COUNTR LESS THAN ZERO
```

where COUNTR is an unsigned numeric item.

A GO TO statement that refers to a previous procedure-name can cause a loop when no conditional statement exists to prevent the GO TO statement from being executed again. For example:

```
PARA-1.  
  MOVE ...  
  MOVE ...  
  MOVE ...  
PARA-2.  
  MOVE ...  
  GO TO PARA-1.
```

A possible variation of this case is when a conditional statement exists, but the condition cannot be met or the statement does not branch (through a GO TO statement) to a paragraph outside the range of the loop.

Recovery after a Failure

Recovery with Commitment Control

When the system is restarted after a failure, files under commitment control are automatically restored to their status at the last commitment boundary.

For a job failure (either because of user or system error), files under commitment control are restored as part of job termination to the files' status at the previous commitment boundary.

Because files under commitment control are rolled back after system or process failure, this feature may be used to aid in restarting. You can create a separate record to store data which may be useful should it become necessary to restart a job. This restart data can include items such as totals, counters, record key values, relative key values, and other relevant processing information from an application.

By having the above restart data in a file under commitment control, that data too will be permanently stored in the data base when a COMMIT statement is issued. When a ROLLBACK occurs after job or process failure, you may retrieve a record of the extent of processing successfully performed before failure. Note that the above method is only a suggested programming technique and will not always be suitable, depending on the application.

Communications Recovery

In some cases, you can recover from I-O errors on TRANSACTION files without intervention by the operator, or the varying off/varying on of work stations or communications devices.

For potentially recoverable I-O errors on TRANSACTION files, the system initiates action in addition to the steps that must be taken in the application program to attempt error recovery. For more information about action taken by the system, see the *CPF Programmer's Guide* and the *Data Communications Programmer's Guide*.

By examining the file status after an I-O operation, the application program can determine whether a recovery from an I-O error on the TRANSACTION file may be possible. If the File Status Key has a value of 9N, the application program may be able to recover from the I-O error. A recovery procedure must be coded as part of the application program and varies depending on whether a single device was acquired by the TRANSACTION file or whether there were multiple devices attached.

For a file with one acquired device:

1. Close the TRANSACTION file with the I-O error.
2. Reopen the file.
3. Perform the steps necessary to retry the failing I-O operation. This may involve a number of steps, depending on the type of program device used.

(For example, if the last I-O operation was a READ, you may have to repeat one or more WRITE statements, which were executed prior to the READ statement.) For more information on recovery procedures, see the *Data Communications Programmer's Guide*.

For a file with multiple devices acquired:

1. DROP the program device that caused the I-O error on the TRANSACTION file.
2. ACQUIRE the same program device.
3. See Step 3 above.

Application program recovery attempts should typically be tried only once.

If the recovery attempt fails:

- If the file has only one program device attached, terminate the program through execution of the STOP RUN or EXIT PROGRAM statement, and attempt to locate the source of the error.
- If the file has multiple acquired program devices, you may wish to do one of the following:
 - Continue with execution without the program device that caused the I-O error on the TRANSACTION file, and reacquire the device later.
 - Terminate the program.

For a description of major-minor return codes that may aid in diagnosing I-O errors on the TRANSACTION file, see the *CPF Programmer's Guide* and the *Data Communications Programmer's Guide*.

Figure 9-17 gives an example of an error recovery procedure.

File		Keying Instruction	Graphic				Description	Page of
Programmer	Date		Key					

Sequence Number	Form Type A=0/1/2/3/4/5/6/7/8/9 N=10/11/12/13/14/15/16/17/18	Conditioning				Name	Length	Reference (R)	Data Type (B/A/P/S/B/F/A/S/X/Y/N/I/M) Decimal Positions Usage (B/O//B/H/M/P)	Location		Functions
		Indicator	Indicator	Indicator	Indicator					Line	Pos	
1	A											
2	A											
3	A											
4	A											
5	A											
6	A											
7	A											
8	A											
9	A											
10	A											
11	A											
12	A											
13	A											
14	A											
15	A											
16	A											
17	A											
18	A											
19	A											
20	A											
21	A											
22	A											
23	A											
24	A											
25	A											
26	A											
27	A											
28	A											
29	A											
30	A											
31	A											
32	A											
33	A											
34	A											
35	A											
36	A											
37	A											
38	A											
39	A											
40	A											
41	A											
42	A											
43	A											
44	A											
45	A											
46	A											
47	A											
48	A											
49	A											
50	A											
51	A											
52	A											
53	A											
54	A											
55	A											
56	A											
57	A											
58	A											
59	A											
60	A											
61	A											
62	A											
63	A											
64	A											
65	A											
66	A											
67	A											
68	A											
69	A											
70	A											
71	A											
72	A											
73	A											
74	A											
75	A											
76	A											
77	A											
78	A											
79	A											
80	A											

*Number of sheets per pad may vary slightly.

Figure 9-17 (Part 1 of 4). Example of Error Recovery Procedure


```

5714CB1 R05 MOD 830610          COBOL SOURCE LISTING          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME

 1 000100 IDENTIFICATION DIVISION.          00010000
 2 000200 PROGRAM-ID. RECOVERY.             00020000
 3 000300* EXAMPLF OF PROGRAM-DEFINED ERROR RECOVERY. 00030000
 4 000400 ENVIRONMENT DIVISION.             00040000
 5 000500 CONFIGURATION SECTION.            00050000
 6 000600 SOURCE-COMPUTER. IBM-538.         00060000
 7 000700 OBJECT-COMPUTER. IBM-538.        00070000
 8 000800 INPUT-OUTPUT SECTION.             00080000
 9 000900 FILE-CONTROL.                     00090000
10 001000 SELECT RECOVFILE                   00100000
11 001100 ASSIGN TO WORKSTATION-RECOVFILE-SI 00110000
12 001200 ORGANIZATION IS TRANSACTION        00120000
13 001300 ACCESS IS SEQUENTIAL               00130000
14 001400 FILE STATUS IS STATUS-FLD, STATUS-FLD-2 00140000
15 001500 CONTROL-AREA IS CTRL-FLD.          00150000
16 001600                                     00160000
17 001700 SELECT PRINTER-FILE                00170000
18 001800 ASSIGN TO PRINTER-QPRINT.          00180000
19 001900                                     00190000
20 002000 DATA DIVISION.                    00200000
21 002100 FILE SECTION.                      00210000
22 002200 FD RECOVFILE                       00220000
23 002300 LABEL RECORDS ARE OMITTED         00230000
24 002400 DATA RECORD IS RECOV-REC.        00240000
25 002500 01 RECOV-REC.                      00250000
26 002600 COPY DDS-ALL-FORMATS OF RECOVFILE. 00260000
27 *000001 05 RECOVFILE-RECORD PIC X(16).    00270000
28 *000002                                     <-ALL-FMTS
29 *000003* INPUT FORMAT:FORMAT1 FROM FILE RECOVFILE OF LIBRARY EXAMPLES <-ALL-FMTS
30 *000004*                                     <-ALL-FMTS
31 *000005 05 FORMAT1-1 REDEFINES RECOVFILE-RECORD. <-ALL-FMTS
32 *000006 06 INPUTFLD PIC X(15).             <-ALL-FMTS
33 *000007                                     <-ALL-FMTS
34 *000008* OUTPUT FORMAT:FORMAT1 FROM FILE RECOVFILE OF LIBRARY EXAMPLES <-ALL-FMTS
35 *000009*                                     <-ALL-FMTS
36 *000010* 05 FORMAT1-0 REDEFINES RECOVFILE-RECORD. <-ALL-FMTS
37 002700                                     00270000
38 002800 FD PRINTER-FILE.                   00280000
39 002900 01 PRINTER-REC.                     00290000
40 003000 05 PRINTER-RECORD PIC X(132).      00300000
41 003100                                     00310000
42 003200 WORKING-STORAGE SECTION.           00320000
43 003300 01 I-O-VERB PIC X(10).             00330000
44 003400 01 STATUS-FLD PIC X(2).            00340000
45 003500 88 NO-ERROR VALUE "00".           00350000
46 003600 88 ACQUIRE-FAILED VALUE "9H".    00360000
47 003700 88 TEMPORARY-ERROR VALUE "9N".    00370000
48 003800 01 STATUS-FLD-2 PIC X(4).          00380000
49 003900 01 CTRL-FLD.                        00390000
50 004000 05 FUNCTION-KEY PIC X(2).          00400000
51 004100 05 PGM-DEVICE-NAME PIC X(10).     00410000
52 004200 05 RECORD-FORMAT PIC X(10).       00420000
53 004300                                     00430000
54 004400 01 END-INDICATOR PIC 1 INDICATOR 1 VALUE "0". 00440000
55 004500 88 END-NOT-REQUESTED VALUF "0".   00450000

```

```

5714CB1 R05 MOD 830610          COBOL SOURCE LISTING          RECOVERY          03/04/83
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME

51 004600 88 END-REQUESTED VALUF "0".        00460000
52 004700 01 USE-PROC-FLAG PIC 1 VALUE "0".  00470000
53 004800 88 USE-PROC-NOT-EXECUTED VALUF "0". 00480000
54 004900 88 USE-PROC-EXECUTED VALUF "1".    00490000
55 005000 01 RECOVERY-FLAG PIC 1 VALUE "0".  00500000
56 005100 88 NO-RECOVERY-DONE VALUF "0".     00510000
57 005200 88 RECOVERY-DONE VALUF "1".        00520000
58 005300                                     00530000
59 005400 01 HEADER-LINE.                     00540000
60 005500 05 FILLER PIC X(160) VALUE SPACES. 00550000
61 005600 05 FILLER PIC X(172) VALUE         00560000
62 005700 "ERROR REPORT".                     00570000
63 005800                                     00580000
64 005900 01 DFTAIL-LINE.                      00590000
65 006000 05 FILLER PIC X(115) VALUE SPACES. 00600000
66 006100 05 DESCRIPTION PIC X(125) VALUE SPACES. 00610000
67 006200 05 DFTAIL-VALUF PIC X(192) VALUE SPACES. 00620000
68 006300                                     00630000
69 006400 01 MESSAGE-LINE.                     00640000
70 006500 05 FILLER PIC X(115) VALUF SPACES. 00650000
71 006600 05 DDESCRIPTION PIC X(117) VALUE SPACES. 00660000

```

Figure 9-17 (Part 2 of 4). Example of Error Recovery Procedure

```

5714C81 R05 M00 830610          COBOL SOURCE LISTING          RECOVERY          03/04/83
STMT SFQNR -A 1 B... 2 ... 3 ... 4 ... 5 ... 5 ... 7 .IDENTFCN S COPYNAME

006700/
006800 PROCEDURE DIVISION.                                0590000
006900 DECLARATIVES.                                       01060000
007000 HANDLE-ERRORS SECTION.
007100     USE AFTER STANDARD ERROR PROCEDURE ON RECOVFILF. 1
007200
007300 DISPLAY-ERROR.
73 007400     SFT USE-PROC-EXECUTED TO TRUE.
74 007500     WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
007600
75 007700     MOVE "ERROR OCCURRED IN" TO DESCRIPTION OF DETAIL-LINE.
76 007800     MOVE I-O-VERB TO DETAIL-VALUE OF DETAIL-LINE.
77 007900     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
008000
78 008100     MOVE "FILE STATUS =" TO DESCRIPTION OF DETAIL-LINE.
79 008200     MOVE STATUS-FLD TO DETAIL-VALUE OF DETAIL-LINE.
80 008300 2 WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 7 LINES.
008400
81 008500     MOVE "EXTENDED FILE STATUS =" TO DESCRIPTION OF DETAIL-LINE.
82 008600     MOVE STATUS-FLD-2 TO DETAIL-VALUE OF DETAIL-LINE.
83 008700     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
008800
84 008900     MOVE "CONTROL AREA =" TO DESCRIPTION OF DETAIL-LINE.
85 009000     MOVE CONTROL-FLD TO DETAIL-VALUE OF DETAIL-LINE.
86 009100     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
009200
009300 CHECK-ERROR.
87 009400     IF TEMPORARY-ERROR AND NO-RECOVERY-DONE THEN
88 009500         MOVE "### ERROR RECOVERY BEING ATTEMPTED ###" TO 3
009600         DESCRIPTION OF MESSAGE-LINE
89 009700         WRITE PRINTER-REC FROM MESSAGE-LINE AFTER ADVANCING
009800         3 LINES
90 009900         PERFORM ERROR-RECOVERY
010000     ELSE
91 010100         IF RECOVERY-DONE THEN
92 010200 4 MOVE "### ERROR AROSE FROM RETRY AFTER RECOVERY ###"
010300         TO DESCRIPTION OF MESSAGE-LINE
93 010400         WRITE PRINTER-REC FROM MESSAGE-LINE AFTER ADVANCING
010500         3 LINES
94 010600         MOVE "### PROGRAM TERMINATED ###" TO DESCRIPTION OF
010700         MESSAGE-LINE
95 010800         WRITE PRINTER-REC FROM MESSAGE-LINE AFTER ADVANCING
010900         2 LINES
96 011000         GO TO ERROR-EXIT
011100     ELSE
97 011200         SET NO-RECOVERY-DONE TO TRUE.
98 011300         MOVE "### EXECUTION CONTINUES ###" TO DESCRIPTION OF
011400         MESSAGE-LINE.
99 011500         WRITE PRINTER-REC FROM MESSAGE-LINE AFTER ADVANCING
011600         2 LINES.
100 011700         GO TO END-OF-DECLARATIVES.
011800
011900 ERROR-RECOVERY.
101 012000     SFT RECOVERY-DONE TO TRUE.
102 012100     DROP PGM-DEVICE-NAME FROM RECOVFILE.

```

```

5714C81 R05 M00 830610          COBOL SOURCE LISTING          RECOVERY          03/04/83
STMT SFQNR -A 1 B... 2 ... 3 ... 4 ... 5 ... 5 ... 7 .IDENTFCN S COPYNAME

103 012200     ACQUIRE PGM-DEVICE-NAME FOR RECOVFILF. 5
012300
012400 ERROR-EXIT.
104 012500     CLOSE RECOVFILE
012600     PRINTER-FILE.
105 012700     STOP RUN.
012800
012900 END-OF-DECLARATIVES.
013000 END DECLARATIVES.

```

Figure 9-17 (Part 3 of 4). Example of Error Recovery Procedure

```

5714CRL R05 M00 830610          COBOL SOURCE LISTING          RECOVERY          03/04/83
SYMT SEQNR  -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN 5 COPYNAME

      013100/
      013200 MAIN-PROGRAM SECTION.
      013300 MAINLINE.
106 013400 MOVE "OPEN" TO I-O-VERB.
107 013500 OPEN I-O RECOVFILE.
      013600 OUTPUT PRINTER-FILE.
108 013700 PERFORM I-U-PARAGRAPH UNTIL END-REQUESTED. 6
109 013800 CLOSE RECOVFILE.
      013900 PRINTER-FILE.
110 014000 STOP RUN.
      014100
      014200 I-U-PARAGRAPH.
111 014300 MOVE "WRITE" TO I-O-VFRR.
112 014400 SET USE-PRJC-NOT-EXECUTED TO TRUE.
113 014500 WRITE RECOV-REC FORMAT IS "FORMAT1"
      014600 INDICATOR IS END-INDICATOR. 7
114 014700 IF USE-PRDC-EXECUTED AND RECOVERY-DONE THEN
115 014800 GO TO I-O-PARAGRAPH.
      014900
116 015000 MOVE "READ" TO I-O-VERB.
117 015100 SET USE-PRJC-NOT-EXECUTED, NO-RECOVERY-DONE TO TRUE.
118 015200 READ RECOVFILE FORMAT IS "FORMAT1"
      015300 INDICATOR IS END-INDICATOR. 8
119 015400 IF NO-ERROR THEN
120 015500 PERFORM SOME-PROCESSING.
      015600
      015700 SOME-PROCESSING.
      015800 (INSERT DATABASE PROCESSING, FOR EXAMPLE)

          * * * * * E N D O F S O U R C E * * * * *

```

- 1 This defines processing that takes place when an I-O error occurs on RECOVFILE.
- 2 This prints out information to help in diagnosing the problem.
- 3 If the file-status equals 9N (temporary error), and no previous error recovery has been attempted for this I-O operation, error recovery is now attempted.
- 4 If recovery was previously attempted, it is not attempted now. This is done to avoid program looping.
- 5 Recovery consists of dropping, then reacquiring, the program device on which the I-O error occurred.
- 6 The mainline of the program consists of writing to and reading from a device until the user signals an end to the program by pressing CF1.
- 7 If the WRITE operation failed but recovery was done, the WRITE is attempted again.
- 8 If the READ operation failed, execution will continue by writing to the device again, and then attempting the READ again.

Figure 9-17 (Part 4 of 4). Example of Error Recovery Procedure

System/38 Inter-Program Communication Considerations

System/38 allows inter-program communication between COBOL and non-COBOL programs. For the following discussion a main program is defined as the COBOL program that is highest in the invocation stack. The main program is the first program in the run unit. A run unit is defined as a set of one or more programs that functions as a unit at execution time to provide a problem solution. A run unit starts with the first COBOL program in the invocation stack and includes all programs (of any type) that are below it in the invocation stack. A subprogram is a program in the run unit below the main program in the invocation stack. For a definition

of invocation stack and other terms relating to System/38 inter-program communication see the *CL Reference Manual*.

Return of Control From a Called Program

In COBOL, you can issue either a `STOP RUN` statement or an `EXIT PROGRAM` statement to return control from a called program. The action of these statements depends upon the execution time environment, as explained below.

- **Exit Program**
 - When issued from a main program, control passes to the next statement (no operation is performed).
 - When issued from a subprogram, control returns to the calling program.
- **Stop Run**
 - Whether issued from a main program or a subprogram, the run unit is terminated. Control returns to the (non-COBOL) program that called the main program.

Initialization of Storage

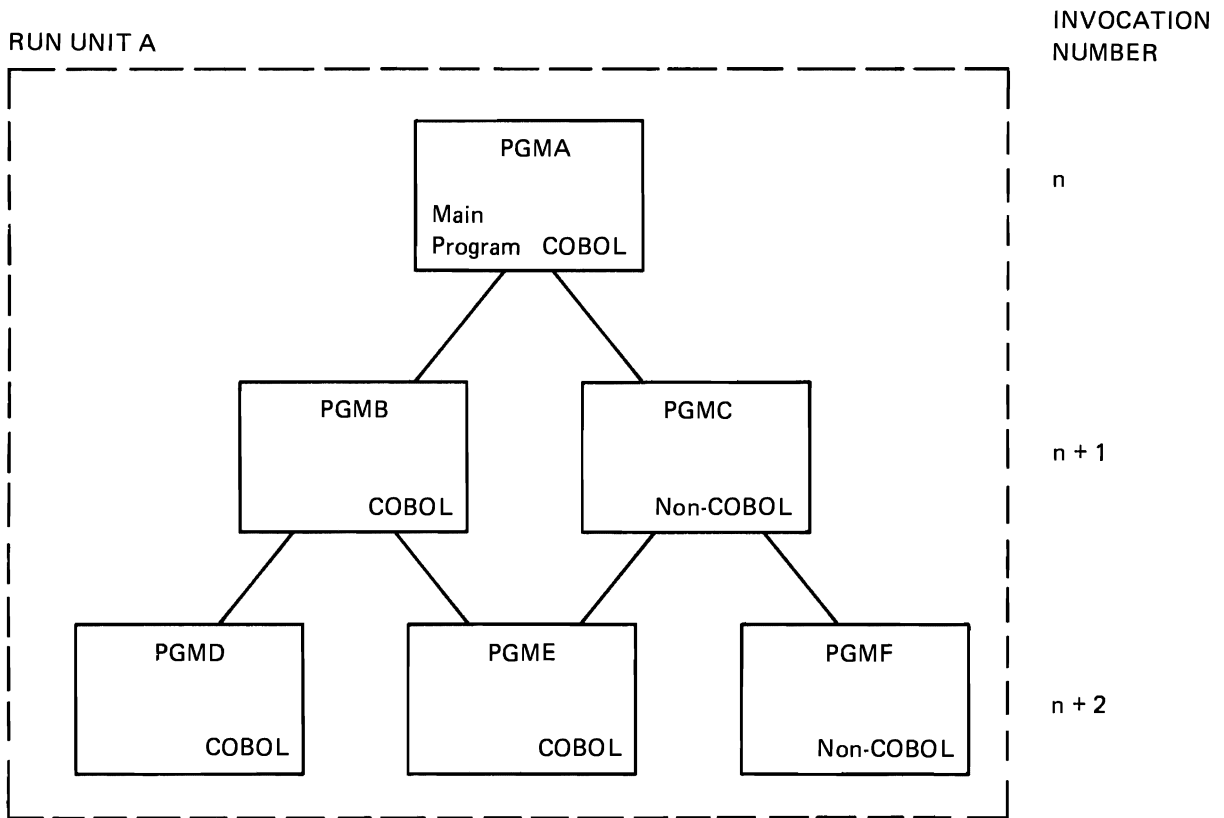
The first time a COBOL program in a run unit is called, its storage is initialized. Storage is initialized again under the following conditions:

- The run unit is terminated, then reinitiated.
- The program is cancelled (`COBOL CANCEL` statement, `RPG III FREE` operation, `CL RCLRSC` command), then called again.

If a non-COBOL program is named in a `CANCEL` statement, its name must conform to the rules for formation of a COBOL program name.

The following examples illustrate the use of the `EXIT PROGRAM` and `STOP RUN` statements in different parts of a run unit.

- The example in Figure 9-18 shows a single run unit.
- The example in Figure 9-19 shows multiple run units.
- The example in Figure 9-20 shows a run unit with a shared program that is both a subprogram and a main program.



PROGRAM EXECUTING STATEMENT

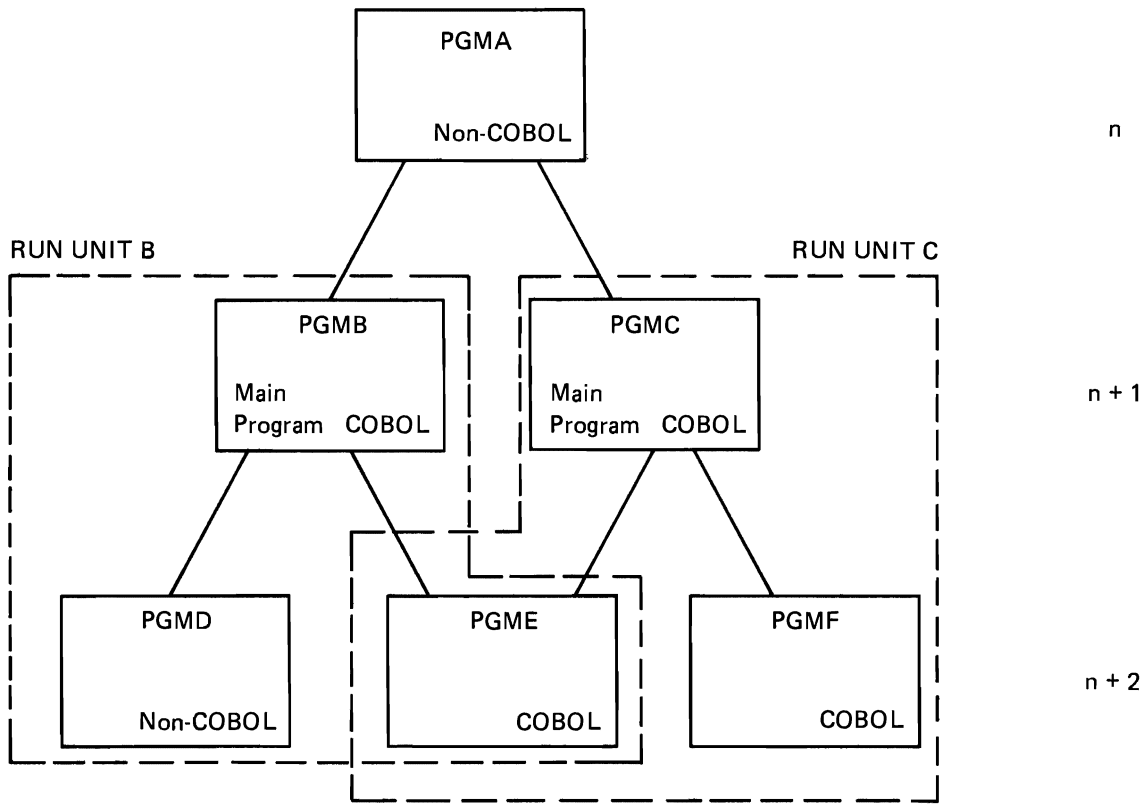
STATEMENT	PGMA	PGMB	PGMD	PGME
EXIT PROGRAM	1	2	2	2
STOP RUN	3	3	3	3

- 1** No operation is performed because the statement is executed in a main program. Execution continues with the next statement in PGMA.
- 2** Control returns to the caller of the program that executes the EXIT PROGRAM statement.
- 3** Run unit A terminates. For all programs in the run unit, open files are closed. Storage is freed for all programs in the run unit. Control returns to the program that is at invocation number n-1. If n=1, the following considerations apply:
 - Run unit A operates as a routing step. See the *CPF Programmer's Guide* for more information.
 - For batch jobs, the STOP RUN causes the job to end. For interactive jobs, control returns to the system and the system terminates the routing step.

Figure 9-18. Example of a Single Run Unit

MULTIPLE RUN UNITS

INVOCATION
NUMBER

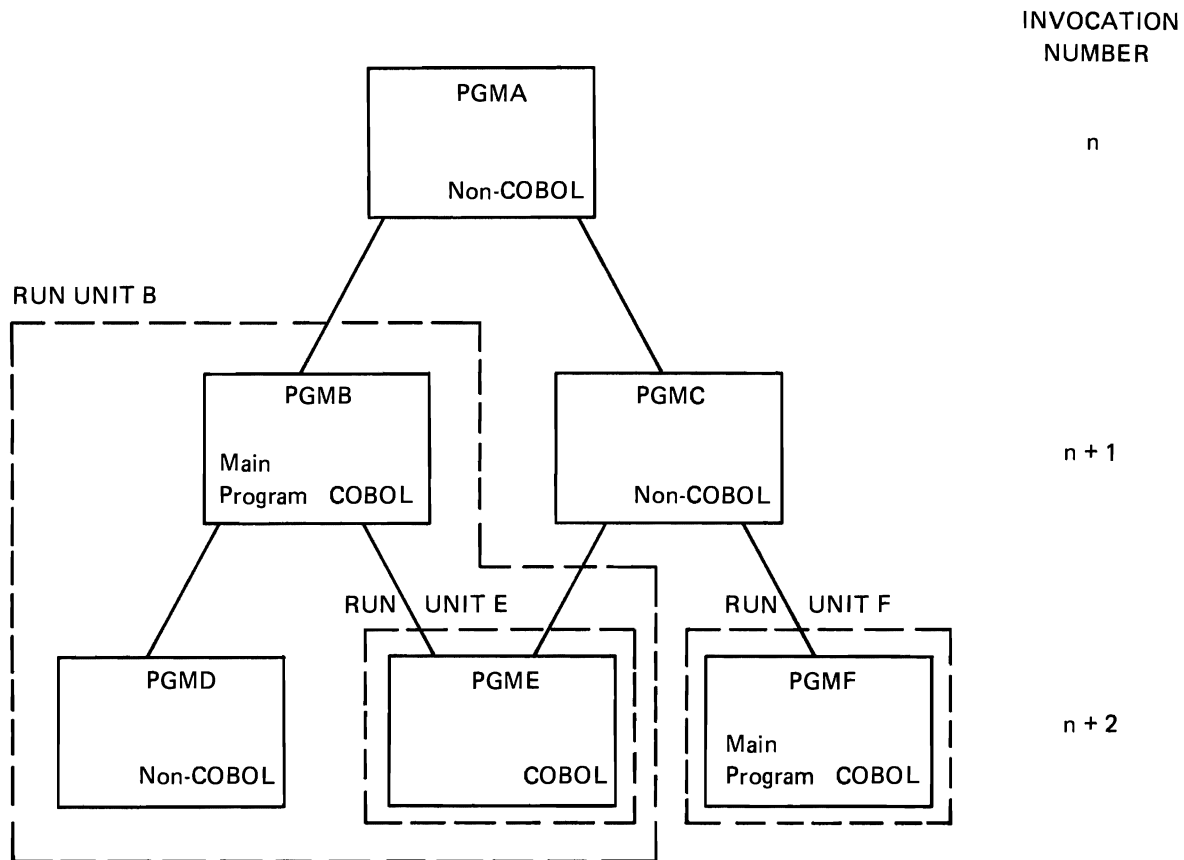


PROGRAM EXECUTING STATEMENT

STATEMENT	PGMB	PGMC	PGME (RUN UNIT B)	PGME (RUN UNIT C)	PGMF
EXIT PROGRAM	1	1	2	2	2
STOP RUN	3	4	3	4	4

- 1** No operation is performed because the statement is executed in a main program. Execution continues with the next statement in the main program.
- 2** Control returns to the caller of the program that executes the EXIT PROGRAM statement.
- 3** Run unit B terminates. All open files in run unit B are closed. Storage is freed for all programs in run unit B. Control returns to the caller of the main program for the run unit (PGMA).
- 4** Run unit C terminates. All open files in run unit C are closed. Storage is freed for all programs in run unit C. Control returns to the caller of the main program for the run unit (PGMA).

Figure 9-19. Example of Multiple Run Units



PROGRAM EXECUTING STATEMENT

STATEMENT	PGMB	PGME (RUN UNIT B)	PGME (RUN UNIT E)	PGMF
EXIT PROGRAM	1	2	1	1
STOP RUN	3	3	4	5

- 1 No operation is performed because the statement is executed in a main program. Execution continues with the next statement in the main program.
- 2 Control returns to the caller of the program that executes the EXIT PROGRAM statement.
- 3 Run unit B terminates. All open files in run unit B are closed. Storage is freed for all programs in run unit B. Control returns to the caller of the main program for the run unit (PGMA).
- 4 Run unit E terminates. All open files in run unit E are closed. Storage is freed for PGME. Control returns to the caller of the main program for the run unit (PGMC).
- 5 Run unit F terminates. All open files in run unit F are closed. Storage is freed for PGMF. Control returns to the caller of the main program for the run unit (PGMC).

Figure 9-20. Example of a Run Unit with a Shared Program that is Both a Subprogram and a Main Program

Local Data Area

The system automatically creates a local data area for each job. The local data area is defined outside the COBOL program as an area of 1024 bytes of character data.

The local data area can be used to pass any desired information between programs in a job. This information may be free-form data, such as informal messages, or may consist of a fully structured or formatted set of fields.

When a job is submitted, the submitting job's local data area is copied into the submitted job's local data area. If there is no submitting job, the local data area is initialized to blanks.

A COBOL program can access the local data area for its job with the ACCEPT and DISPLAY statements, using a mnemonic name associated with the function-name LOCAL-DATA.

There is only one local data area associated with each job. If several work stations are acquired by a single job, still only one local data area exists for that job. There is *not* a local data area for each individual work station.

File Considerations

A file cannot be received as a parameter in a COBOL program. If a file is defined in both a calling program and a called program, it is treated as two separate files. The contents of the record area and the current record pointer in each program are independent, unless shared files are specified in CL commands. See the *CL Reference Manual* for further information on shared files.

The following statements affect file status differently:

- An EXIT PROGRAM statement does not change the status of any of the files in a run unit.
- A STOP RUN statement closes all the files in a run unit.
- A CANCEL statement does not change the status of any of the files in the program that is canceled. It does free the storage that contains information about the file. If the program has files that are open when the CANCEL statement is executed, those files remain open until the run unit is terminated. The program can no longer use the file. If the cancelled program is called again, the program considers the file closed. If the program opens the file, a new linkage to the file is established. This can cause additional system storage to be used.



Chapter 10. Testing and Debugging COBOL Programs

COBOL and CPF provide functions that you can use to test and debug the programs you develop:

CPF	COBOL
Test library	Debugging features
Breakpoints	Formatted dump
Traces	

The CPF functions let you test programs while protecting your production files, and let you observe and debug operations as a program executes. No special source code is required for using the CPF functions.

The COBOL functions can be used independently of the CPF functions or in combinations with them to:

- Debug a program
- Produce a formatted dump of the contents of fields, data structures, arrays, and tables.

Source code is required for using the COBOL Debugging features and formatted dump capability. A formatted dump can also be obtained by a user's response to an execution-time message.

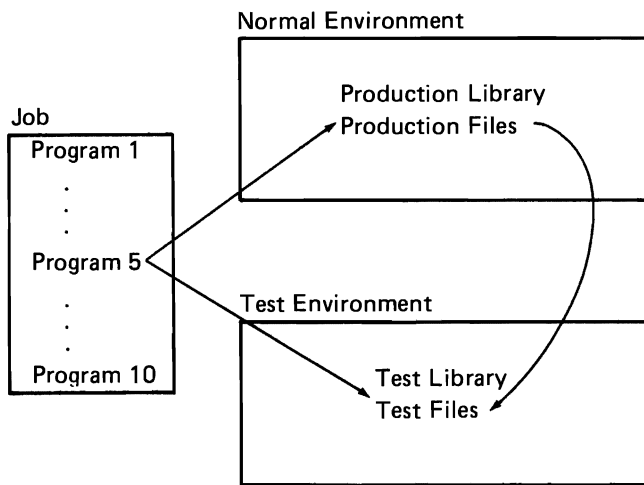
OPEN-FEEDBACK and I-O-FEEDBACK contents can provide additional debugging information. The method for obtaining this information is described later in this chapter.

Using a Test Library

The basic concept of testing and debugging is that of a testing environment. Programs executing in a normal operating environment can read, update, and write records that are in either test or production libraries. Programs executing in a test environment can read, update, and write records in either test or production libraries. However, to prevent data base files in production libraries from being modified unintentionally, you can specify UPDPROD(*NO) on the Enter Debug (ENTDBG) command or Change Debug (CHGDBG) command (see the *CPF Programmer's Guide*).

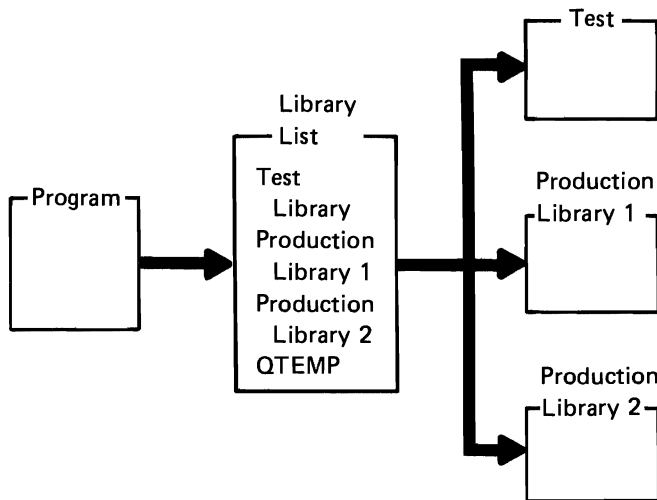
On System/38, you can copy production files into the test library or you can create special files for testing in this library. A test copy of a file and its

production copy can have the same name if the files are in different libraries. You can use the same file name in the program for either testing or normal processing.

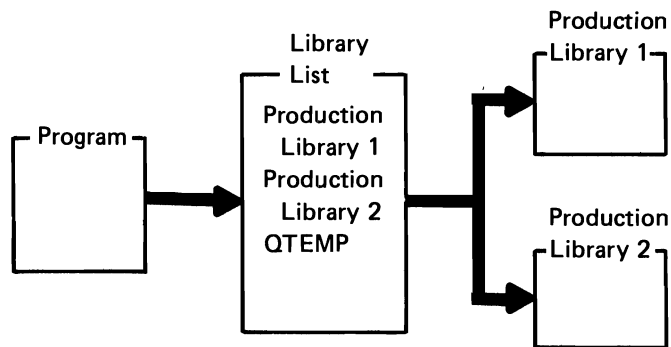


For testing, you must place the test library name ahead of the production library name in the library list for the job that contains the program to be tested. For normal execution, the test library should not be named in the library list for that job.

Testing

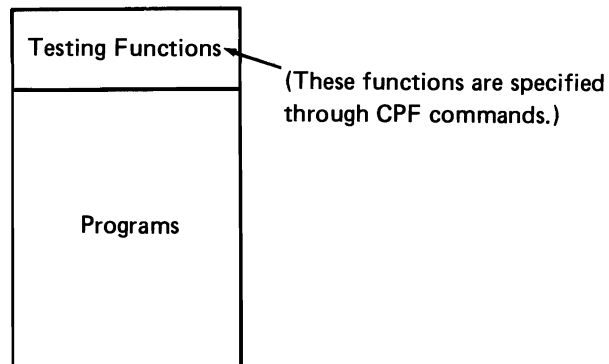


Normal



No special statements for testing are contained within the program being tested. The same program being tested can be run normally without modifications. All testing functions are specified within the job that contains the program and not within the program.

Job



Testing functions apply only to the job in which they are specified. A program can be used concurrently in two jobs: one job that is in a test environment and another job that is in a normal processing environment.

Testing functions of CPF let you interact with a program as it executes to observe the operations being done. These functions include using breakpoints and traces.

Using Breakpoints

A breakpoint is a statement number or a label in your program where you want program execution to stop. If you use a statement number, it can be a statement number that appears on the compiler listing of the COBOL source program. If you use a label as a breakpoint rather than a statement number, the label can be:

- Associated with a function done by your COBOL program (for example, .OPEN indicates the open file function). See “Examples of Using Compiler Debugging Options” in Appendix A for a list of breakpoints and their functions.
- An internal COBOL compiler generated label (for example, .L000001 indicates the first internally generated label).

Note: To determine the internally generated labels for your program, use the GENOPT parameter on the CRTCLPGM command to get an IRP listing of the program.

When a breakpointed statement is about to execute for an interactive job, the system displays the breakpoint at which the program has stopped and, if requested, the values of program variables. After you get this information (in a display), you can enter CPF commands to request other functions (such as displaying or changing a variable, adding a breakpoint, or adding a trace).

For a batch job, a breakpoint program can be invoked when a breakpoint is reached. The breakpoint information is passed to the breakpoint program.

Example of Using Breakpoints

Figure 10-1 shows a sample COBOL program, TESTPRT. The following CPF commands add breakpoints at statements 51 and 60. The value of variable RECORDNO is displayed when the breakpoint at statement 60 is reached.

CPF Commands:

```
ENTDBG  TESTPRT
ADDBKP  STMT(51)
ADDBKP  STMT(60)
        PGMVAR(RECORDNO)
```

All CPF commands are explained in the *CL Reference Manual*.

```

                                COBOL SOURCE LISTING
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME  CHG/DATE

 2 000200 IDENTIFICATION DIVISION.
 3 000300 PROGRAM-ID. TESTPRT.
 4 000400 AUTHOR. PROGRAMMER NAME.
 5 000500 INSTALLATION. ROCHESTER LABORATORY.
 6 000600 DATE-WRITTEN. JULY 15, 1980.
 7 000700 DATE-COMPILED. 11/06/80 18:32:31 .
 8 000800 ENVIRONMENT DIVISION.
 9 000900 CONFIGURATION SECTION.
10 001000 SOURCE-COMPUTER. IBM-538.
11 001100 OBJECT-COMPUTER. IBM-538.
12 001200 INPUT-OUTPUT SECTION.
13 001300 FILE-CONTROL.
14 001400     SELECT FILE-1 ASSIGN TO DATABASE-MASTER.
15 001500     SELECT FILE-2 ASSIGN TO DATABASE-MASTER.
16 001600 DATA DIVISION.
17 001700 FILE SECTION.
18 001800 FD FILE-1
19 001900     LABEL RECORDS ARE STANDARD
20 002000     RECORD CONTAINS 20 CHARACTERS
21 002100     DATA RECORD IS RECORD1.
22 002200 01 RECORD1.
23 002300 02 FIELDA PICTURE IS X(20).
24 002400 FD FILE-2
25 002500     LABEL RECORDS ARE STANDARD
26 002600     RECORD CONTAINS 20 CHARACTERS
27 002700     DATA RECORD IS RECORD2.
28 002800 01 RECORD2.
29 002900 02 FIELDA PICTURE IS X(20).
30 003000 WORKING-STORAGE SECTION.
31 003100 01 FILLER.
32 003200 05 KOUNT PIC S99 COMP-3.
33 003300 05 ALPHABET PICTURE X(26) VALUE "ABCDEFGHJKLMNOPQRSTUVWXYZ".
34 003400 05 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
35 003500 05 NUMBR PIC S99 COMP-3.
36 003600 05 NO-OF-DEPENDENTS PIC X(26)
37 003700     VALUE "01000012000123001234002340".
38 003800 05 DEPEND REDEFINES NO-OF-DEPENDENTS PIC X OCCURS 26 TIMES.
39 003900 01 WORKRECORD.
40 004000 05 NAMES-FIELD PICTURE X.
41 004100 05 FILLER PICTURE X VALUE IS SPACE.
42 004200 05 RECORDNO PICTURE S999.
43 004300 05 FILLER PICTURE X VALUE IS SPACE.
44 004400 05 LOCATION PICTURE AAA VALUE IS "NYC".
45 004500 05 FILLER PICTURE X VALUE IS SPACE.
46 004600 05 DEPENDENTS PICTURE XX.
47 004700 05 FILLER PICTURE X(7) VALUE IS SPACES.
48 004800 01 NODEPEND PICTURE 99 COMP-3.
49 004900 01 ZONE-DATA PICTURE 99.
50 005000 PROCEDURE DIVISION.
005100*****
005200** THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO BE CREATED**
005300** AND INITIALIZES COUNTERS. **
005400*****
005500 STEP-1.
51 005600     OPEN OUTPUT FILE-1.

```

```

                                COBOL SOURCE LISTING                                TESTPRT
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME  CHG/DATE

52 005700     MOVE ZERO TO KOUNT NUMBR NODEPEND.
005800*****
005900** THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE **
006000** CONTAINED IN THE FILE. WRITES THEM ON DISK, AND DISPLAYS **
006100** THEM. **
006200*****
006300 STEP-2.
53 006400     ADD 1 TO KOUNT. ADD 1 TO NUMBR.
54 006500     MOVE ALPHA (KOUNT) TO NAMES-FIELD.
55 006600     MOVE DEPEND (KOUNT) TO DEPENDENTS.
56 006700     MOVE NUMBR TO RECORDNO.
006800 STEP-3.
57 006900     WRITE RECORD1 FROM WORKRECORD.
007000 STEP-4.
58 007100     PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
007200*****
007300** THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS **
007400** INPUT. **
007500*****
007600 STEP-5.
59 007700     CLOSE FILE-1.
60 007800     OPEN INPUT FILE-2.
61 007900*****
008000** THE FOLLOWING READS BACK THE FILE AND SINGLES OUT EMPLOYEES**
008100** WITH NO DEPENDENTS. **
008200*****
008300 STEP-6.
62 008400     READ FILE-2 RECORD INTO WORKRECORD
63 008500     AT END GO TO STEP-8.
008600 STEP-7.
64 008700     IF DEPENDENTS IS EQUAL TO "0"
65 008800     MOVE "Z" TO DEPENDENTS
66 008900     ADD 1 TO NODEPEND.
67 009000     GO TO STEP-6.
009100 STEP-8.
68 009200     CLOSE FILE-2.
69 009300     DISPLAY "EMPLOYEES WITH NO DEPENDENTS " NODEPEND.
70 009400     ADD 1 TO ZONE-DATA.
71 009500     STOP RUN.

***** END OF SOURCE *****

```

Figure 10-1. Example of Using Breakpoints

The first breakpoint is used just so you know where you are in the program. The following is displayed as a result of reaching the first breakpoint.

```
06/16/81 11:08:31   BREAKPOINT DISPLAY
Stmt/Inst:  51      /0015
Program:    TESTPRT   Inv lvl:  1

CF1-Cancel request  CF3-Command Entry  ENTER-Resume execution
```

The following is displayed as a result of reaching the second breakpoint.

```
06/16/81 11:09:54   BREAKPOINT DISPLAY
Stmt/Inst:  60      /003A
Program:    TESTPRT   Inv lvl:  1
Output start pos: 1   length:   *DCL   format: *CHAR
Variable:   RECORDNO
  Type:     ZONED           Length:    3 0
  '+26'
```

```
CF1-Cancel request  CF3-Command Entry  ENTER-Resume execution
```

When specifying a variable for the PGMVAR parameter, every name must begin with an alphanumeric character (A through Z, \$, #, or @) and can be followed by the characters (A through Z, 0 through 9, \$, #, @, or _).

The following example shows how to display a COBOL variable, ZONE-DATA, in the sample program. Because the hyphen is treated by CPF as a special character, ZONE-DATA must be enclosed in quotes.

```
ENTDBG  TESTPRT
ADDBKP  STMT(60)
        PGMVAR('ZONE-DATA')
```

To display the value of a table element, the appropriate occurrence numbers (subscripts) must be included with the variable name. Up to three dimensions of subscripting are allowed, and the subscripts must be separated by commas. You should not use an index-name or index data-item as a subscript. When an index is entered as a subscript, CPF uses the internal value of the index as the subscript, and undesirable results can occur. The following example shows how to specify the COBOL variable TABLE1 with three dimensions.

```
PGMVAR ('TABLE1 (SUB1, SUB2, SUB3)').
```

One or more blanks are allowed after each comma separating subscripts, but the total length of the variable plus subscripts, parentheses, commas, and blanks specified with the PGMVAR keyword cannot exceed 132 characters. For more information on how to code variables in CL commands, see the *CL Reference Manual*.

Variable names can be qualified in the PGMVAR parameter, for example, NAMES-FIELD OF WORKRECORD — Figure 10-1.

Another technique can be used to display variables that are not elements of a multi-dimensional table. For example, to display the field NAMES-FIELD you can use COBOL Data Division map to find its COBOL internal name (I-NAME). Next use the IRP cross-reference listing to find the Object Definition Table (ODT) number for the internal-name. (See “COBOL Command Statement” and “PROCESS Statement” in Chapter 8 for information on how to obtain these listings.) Figure 10-2 shows the Data Division map and Figure 10-3 shows the IRP cross-reference listing for the sample program, TESTPRT.

COBOL DATA DIVISION MAP						TESTPRT		
STMT	LVL	SOURCE NAME	SECTION	DISP	LEN	TYPE	I-NAME	ATTRIBUTES
18	FD	FILE-1	FS				.F01	DEVICE DATABASE, ORGANIZATION SEQUENTIAL, ACCESS SEQUENTIAL, BLOCK CONTAINS 20 CHARACTERS, RECORD CONTAINS 20 CHARACTERS, LABEL RECORDS STANDARD
22	01	RECORD1	FS	000000	20	GROUP	.D005394	
23	02	FIELDA	FS	000000	20	AN	.D0053E8	
24	FD	FILE-2	FS				.F02	DEVICE DATABASE, ORGANIZATION SEQUENTIAL, ACCESS SEQUENTIAL, BLOCK CONTAINS 20 CHARACTERS, RECORD CONTAINS 20 CHARACTERS, LABEL RECORDS STANDARD
28	01	RECORD2	FS	000000	20	GROUP	.D0054E2	
29	02	FIELDA	FS	000000	20	AN	.D005536	
31	01	FILLER	WS	000000	56	GROUP	.D005580	
32	02	KOUNT	WS	000000	2	PACKED	.D0055D2	
33	02	ALPHABET	WS	000002	26	AN	.D005634	VALUE
34	02	ALPHA	WS	000002	1	AN	.D005680	REDEFINES .D005634, DIMENSION(26)
35	02	NUMBR	WS	000028	2	PACKED	.D005710	
36	02	NO-OF-DEPENDENTS	WS	000030	26	AN	.D005772	VALUE
38	02	DEPEND	WS	000030	1	AN	.D0057F6	REDEFINES .D005772, DIMENSION(26)
39	01	WORKRECORD	WS	000000	19	GROUP	.D005856	
40	02	<u>NAMES-FIELD</u>	WS	000000	1	AN	<u>.D0058AC</u>	
41	02	FILLER	WS	000001	1	AN	.D005904	VALUE
42	02	RECORDNO	WS	000002	3	ZONED	.D00595E	
43	02	FILLER	WS	000005	1	AN	.D0059C2	VALUE
44	02	LOCATION	WS	000006	3	A	.D005A1C	VALUE
45	02	FILLER	WS	000009	1	AN	.D005A82	VALUE
46	02	DEPENDENTS	WS	000010	2	AN	.D005ADC	
47	02	FILLER	WS	000012	7	AN	.D005B32	VALUE
48	01	NODEPEND	WS	000000	2	PACKED	.D005B8C	
49	01	ZONE-DATA	WS	000000	2	ZONED	.D005BEE	
14		DB-FORMAT-NAME	SR	000079	10	AN	.D005CAA	

FILE SECTION USES 40 BYTES OF STORAGE
WORKING-STORAGE SECTION USES 89 BYTES OF STORAGE

Figure 10-2. Data Division Map for TESTPRT

IBM S/38 COBOL 5714C01		IRP LISTING FOR TESTPRT	
ODT	ODT NAME	SEQ CROSS REFERENCE (* INDICATES WHERE DEFINED)	
01A8	.D005502	547*	602 607 611 615 637
01A9	.D005536	545*	
01AA	.D005580	546*	547 548 551 552
01AE	.D005680	550*	613
01AC	.D005634	548*	549
01B2	.D005776	554*	617
01AF	.D005710	551*	603 609 619
01B0	.D005772	552*	553
01B3	.D005896	555*	556 557 558 559 560 561 562 563 622 678
01B7	.D0059C2	559*	
01B5	.D005904	557*	
01B6	.D00595E	558*	619
01C0	.D00707C	568*	709
0224	.EXCALLX	879*	
0211	.EXCODE	838*	881
0223	.EXFSGN	878*	886
0216	.EXFS10	844*	851
021C	.EXFS22	864*	866
021E	.EXFS34	868*	870
0220	.EXFS9K	872*	874
0226	.EXGNUSE	881	882 884*
0212	.EXMSG10	839*	882
0219	.EXNDP	853*	862
021A	.EXOPEDF	854	860*
0214	.EXOPLST	841*	857 880
0210	.EXPARMS	837*	838 839 840 856 879
0213	.EXPTR	840*	841
00B0	.FCEXCM	181*	
00BC	.FCLIST	180*	182
00D0	.FCLDP	286*	
00CE	.FCLSTC	284*	285 286
00CD	.FCLSTC#	283*	
00D2	.FCLSTP	289*	290 291 292 293
00D1	.FCLSTP#	288*	
00BA	.FCPARM	178*	179
00BB	.FCPARMP	179*	180
00B9	.FCPTR	177*	182
0094	.FIB	196*	
00C2	.FIB#OPT	269*	270 271 272 273 274
0096	.FIBALT	198*	595 759
00AE	.FIBCA	248*	
00B1	.FIBCFMT	251*	
009D	.FIBCFPS	222*	527 628 747 749 753 764 774 793 795 810 829 847 849 855 864 868 872 878 883
009E	.FIBCFPS1	223*	672
00A7	.FIBCHAN	239*	
00AF	.FIBCKID	249*	
009C	.FIBGCP	221*	792 815 818 828 832
00B0	.FIBCTID	250*	
0113	.FIBCTL	371*	
0098	.FIBCUR	220*	221 222 223 746 791 809 827
00C8	.FIBCURK	275*	276 277
0097	.FIBFLGS	208*	748 752 756 769 784 787 794 802 803 845 846 854 860
00A0	.FIBFMT	225*	
0095	.FIBFN	197*	
00C4	.FIBKMLN	271*	

Figure 10-3. IRP Cross-Reference Listing for TESTPRT

You can use the following CPF commands to add a breakpoint to the sample program at statement 60 that will display the variable NAMES-FIELD, using the appropriate ODT number:

```

ENTDBG TESTPRT
ADDBKP STMT(60)
PGMVAR('/odt-number')

```

This command is explained in the *CL Reference Manual*.

The following is displayed as a result of reaching this breakpoint:

```

06/16/81 11:08:31 BREAKPOINT DISPLAY
Stmt/Inst: 60 /003A
Program: TESTPRT Inv lvl: 1
Output start pos: 1 length: *DCL format: *CHAR
Variable: /01B4
Type: CHARACTER Length: 1
'Z'

CF1-Cancel request CF3-Command Entry ENTER-Resume execution

```

At this point you could change the value of program variables to alter your program's execution. You can use the Change Program Variable (CHGPGMVAR) command to change the value of a variable. in the *CL Reference Manual*.

Considerations for Using Breakpoints

The following characteristics of breakpoints should be known before breakpoints are used:

- If a breakpoint is bypassed by a COBOL statement, for example the GO TO statement, that breakpoint does not execute.
- When a breakpoint is established on a statement, the breakpoint occurs before that statement is executed.
- Breakpoint functions are specified through CPF commands.

These functions include adding breakpoints to programs, displaying breakpoint information, removing breakpoints from programs and resuming execution of a program after a breakpoint has been displayed. See the *CL Reference Manual* for descriptions of these commands, and see the *CPF Programmer's Guide* for a further description of breakpoints.

Using a Trace

A trace is a record of some or all of the statements in a program that were executed and, if requested, the values of specific variables used in the statements.

Program	Trace	
Statement	Order of Execution	Variables
1 ~~~~~	1 →	~~~~~
2 ~~~~~	6 →	~~~~~
3 ~~~~~	7 →	~~~~~
4 ~~~~~	8 →	~~~~~
5 ~~~~~	6 →	~~~~~
6 ~~~~~	7 →	~~~~~
7 ~~~~~	2 →	~~~~~
8 ~~~~~	6 →	~~~~~
.	7 →	~~~~~
.	.	.
.	.	.

A trace differs from a breakpoint in that a trace ends depending on which statements and how many statements are traced. The system records the traced statements that were executed. You must request a display of the traced information. The display shows the sequence in which the statements were executed and, if requested, the values of variables used in the statements.

You specify what statements the system should trace. Also, you might specify that variables be displayed only when their value changes from the previous time a traced statement executed.

You can specify a trace of one statement in a program, a group of statements in a program, or an entire program.

Example of Using a Trace

Figure 10-4 shows a portion of a sample COBOL program, TESTPRT. The following CPF command adds a trace of statements 62 and 67 in that program. The variable NODEPEND is to be recorded only if its value changes between statements 62 and 67:

```
ADDTRC STMT (62 67)
        PGMVAR(NODEPEND) OUTVAR(*CHG)
```

Note: ENTDBG must be entered before the ADDTRC statement.

```

                                COBOL SOURCE LISTING          TESTPRT
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN 5  COPYNAME  CHG/DATE

50 005000 PROCEDURE DIVISION.
005100*****
005200** THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO BE CREATED**
005300** AND INITIALIZES COUNTERS.          ***
005400*****
005500 STEP-1.
51 005600 OPEN OUTPUT FILE-1.
52 005700 MOVE ZERO TO KOUNT NUMBR NODEPEND.
005800*****
005900** THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
006000** CONTAINED IN THE FILE. WRITES THEM ON DISK, AND DISPLAYS
006100** THEM.          ***
006200*****
A 006300 STEP-2.
53 006400 ADD 1 TO KOUNT. ADD 1 TO NUMBR.
55 006500 MOVE ALPHA (KOUNT) TO NAMES-FIELD.
56 006600 MOVE DEPEND (KOUNT) TO DEPENDENTS.
57 006700 MOVE NUMBR TO RECORDNO.
006800 STEP-3.
58 006900 WRITE RECOND1 FROM WORKRECORD.
007000 STEP-4.
59 007100 PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
007200*****
007300** THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
007400** INPUT.          ***
007500*****
007600 STEP-5.
60 007700 CLOSE FILE-1.
61 007800 OPEN INPUT FILE-2.
007900*****
008000** THE FOLLOWING READS BACK THE FILE AND SINGLES OUT EMPLOYEES**
008100** WITH NO DEPENDENTS.          ***
008200*****
008300 STEP-6.
62 008400 READ FILE-2 RECORD INTO WORKRECORD
008500 AT END GO TO STEP-8.
008600 STEP-7.
64 008700 IF DEPENDENTS IS EQUAL TO "0"
65 008800 MOVE "Z" TO DEPENDENTS
008900 ADD 1 TO NODEPEND.
67 009000 GO TO STEP-6.
009100 STEP-8.
68 009200 CLOSE FILE-2.
69 009300 DISPLAY "EMPLOYEES WITH NO DEPENDENTS " NODEPEND.
70 009400 ADD 1 TO ZONE-DATA.
71 009500 STOP RUN.

```

Figure 10-4. COBOL Source Code

Figure 10-5 is an example of a display of the traced information. The following CPF command lists this information:

```
DSPTRCDTA OUTPUT(*LIST) CLEAR(*YES)
```

The Display Trace Data command is explained in the *CPF Reference Manual - CL*.

11/06/80 19:30:57		TRACE DATA DISPLAY		
STMT: 62	PGM: TESTPRT	LVL: 1		54
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 40*	PACKED	LENGTH: 2 0		
STMT: 67	PGM: TESTPRT	LVL: 1		55
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 41*	PACKED	LENGTH: 2 0		
STMT: 62	PGM: TESTPRT	LVL: 1		56
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 41*	PACKED	LENGTH: 2 0		
STMT: 67	PGM: TESTPRT	LVL: 1		57
STMT: 62	PGM: TESTPRT	LVL: 1		58
STMT: 67	PGM: TESTPRT	LVL: 1		59
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 42*	PACKED	LENGTH: 2 0		
STMT: 62	PGM: TESTPRT	LVL: 1		60
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 42*	PACKED	LENGTH: 2 0		
STMT: 67	PGM: TESTPRT	LVL: 1		61
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 43*	PACKED	LENGTH: 2 0		
STMT: 62	PGM: TESTPRT	LVL: 1		62
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 43*	PACKED	LENGTH: 2 0		
STMT: 67	PGM: TESTPRT	LVL: 1		63
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 44*	PACKED	LENGTH: 2 0		
STMT: 62	PGM: TESTPRT	LVL: 1		64
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 44*	PACKED	LENGTH: 2 0		
STMT: 67	PGM: TESTPRT	LVL: 1		65
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 45*	PACKED	LENGTH: 2 0		
STMT: 62	PGM: TESTPRT	LVL: 1		66
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		
*VARIABLE: NODEPEND				
* 45*	PACKED	LENGTH: 2 0		
STMT: 67	PGM: TESTPRT	LVL: 1		67
STMT: 62	PGM: TESTPRT	LVL: 1		68
STMT: 67	PGM: TESTPRT	LVL: 1		69
STMT: 62	PGM: TESTPRT	LVL: 1		70
STMT: 67	PGM: TESTPRT	LVL: 1		71
OUTPUT START POS: 1	LENGTH: #DCL	FORMAT: #CHAR		

Figure 10-5. Trace Data Display Listing

Considerations for Using a Trace

The following characteristics of traces should be known before traces are used in COBOL programs:

- Statements bypassed by COBOL statements, such as the GO TO statement, are not included in the trace.
- Trace functions are specified through CPF commands in the job that contains the traced program. These functions include adding trace requests to a program, removing trace requests from a program, removing data collected from previous traces, displaying trace information, and displaying the traces that have been specified for a program.
- In addition to statement numbers, names of COBOL-generated routines can appear on the trace output STMT field. (See the list of breakpoints and their functions under “Examples of Using Compiler Debugging Options” in Appendix A.)

See the *CPF Programmer's Guide* for a further description of traces.

Using a Debug Execution-Time Switch

An execution-time switch is provided for the COBOL Debug facility. This switch activates the debugging code generated when WITH DEBUGGING MODE is specified. When the execution-time switch is set on, all debugging sections that were compiled are activated. When the execution-time switch is set off (the default), the USE FOR DEBUGGING Declarative procedures are inhibited. In both cases the debugging lines (D in column 7) remain in effect.

See “Debugging Features” in Chapter 6 for more information on COBOL Debug and how to use the switch.

File Status

The following format can be used to transfer data (OPEN-FEEDBACK or I-O-FEEDBACK areas) associated with an open file to an identifier.

Format

```
ACCEPT identifier FROM mnemonic-name  
[ FOR file-name ]
```

See Chapter 5 for more information on specifying this statement.

Appendix H contains a discussion of the OPEN-FEEDBACK and I-O-FEEDBACK areas. See the *CPF Programmer's Guide* for a layout and description of the data areas contained in the feedback areas.

Using a COBOL Formatted Dump

Some COBOL execution-time messages allow you to obtain a COBOL formatted dump option by giving a reply of D or F.

The output for the dump is sent to the IBM-supplied printer file QPPGMDMP.

The formatted dump (reply D) includes the current file information about the files in the program, contents of fields, data structures, arrays, and tables for user-defined COBOL data variables.

If you reply with an F option, the dump also includes a list of compiler-generated fields and their contents.

If you do not desire a dump, specify reply C (cancel with *no* dump). Reply C is also the default reply for all COBOL inquiry messages that allow a dump to be obtained.

Reply Modes and System Reply List

The need for a reply to execution time inquiry messages is controlled by the INQMSGRPY parameter on the following control language (CL) commands:

- CHGJOB — Change Job
- CHGJOB — Change Job Description
- CRTJOB — Create Job Description
- JOB — Job
- SBMJOB — Submit Job.

Through the INQMSGRPY parameter, three reply modes for inquiry messages are possible. The reply modes are:

- *RQD — A reply is required.
- *DFT — Take the default reply.
- *SYSRPL — Search the System Reply List and if the message is found, take the default action specified in the list. If the message is not found in the System Reply List, then take the default specified in the message description.

The System Reply List allows you to specify replies for inquiry messages CBE7200, CBE7201, CBE7203, CBE7204, and CBE7205. The replies may be specified individually or generically. This method of replying to inquiry messages is especially suitable for batch programs, which would otherwise require a console operator to issue replies.

To see the entries in the System Reply List, specify the Display Reply List (DSPRPLY) CL command. This allows you to change, remove, and add entries to the System Reply List.

For more information on message reply modes and the System Reply List, see the *CL Reference Manual*.

Example of Using a Dump

Figure 10-6 shows an example of a COBOL formatted dump.

The following list describes the areas of Figure 10-6 indicated by letters:

- A** The exception for which the dump was requested and the location in the program where the exception occurred.
- B** The COBOL statement number of the last I-O operation that was executed before the exception occurred. This information is produced only if at least one I-O operation has been executed.
- C** Current information for each file. This information is produced only if the program has files.
- D** Beginning of compiler-generated fields (included in the dump if the user responds with an F option).
- E** I-O flags for the current file:

Bit Meaning

1	File is open
2	File is locked
3	End of file
4	MFCU
5	Optional file
6	Check indexed file for duplicates at open
7	End of page
8	(Reserved)

- F** Previous status code.
- G** Beginning of Module Global Table (MGT).
- H** Last exception code.
- I** Invocation number of current program.
- J** Qualified program name and library.
- K** Beginning of the Program Global Table (PGT).
- L** Invocation number of the main COBOL program.
- M** Job date (YYMMDD).
- N** Beginning of user fields.
- O** Invalid zoned field printed in hexadecimal.

```

                                COBOL SOURCE LISTING
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S COPYNAME  CHG/DATE

 2 000200 IDENTIFICATION DIVISION.
 3 000300 PROGRAM-ID, TESTPRT.
 4 000400 AUTHOR, PROGRAMMER NAME.
 5 000500 INSTALLATION, ROCHESTER LABORATORY.
 6 000600 DATE-WRITTEN, JULY 15, 1980.
 7 000700 DATE-COMPILED, 11/06/80 16:32:31 .
 8 000800 ENVIRONMENT DIVISION.
 9 000900 CONFIGURATION SECTION.
10 001000 SOURCE-COMPUTER, IBM-S38.
11 001100 OBJECT-COMPUTER, IBM-S38.
12 001200 INPUT-OUTPUT SECTION.
13 001300 FILE-CONTROL.
14 001400     SELECT FILE-1 ASSIGN TO DATABASE-MASTER.
15 001500     SELECT FILE-2 ASSIGN TO DATABASE-MASTER.
16 001600 DATA DIVISION.
17 001700 FILE SECTION.
18 001800 FD FILE-1
19 001900     LABEL RECORDS ARE STANDARD
20 002000     RECORD CONTAINS 20 CHARACTERS
21 002100     DATA RECORD IS RECORD1.
22 002200 01 RECORD1.
23 002300 02 FIELDA PICTURE IS X(20).
24 002400 FD FILE-2
25 002500     LABEL RECORDS ARE STANDARD
26 002600     RECORD CONTAINS 20 CHARACTERS
27 002700     DATA RECORD IS RECORD2.
28 002800 01 RECORD2.
29 002900 02 FIELDB PICTURE IS X(20).
30 003000 WORKING-STORAGE SECTION.
31 003100 01 FILLER.
32 003200 05 KOUNT PIC S99 COMP-3.
33 003300 05 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
34 003400 05 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
35 003500 05 NUMBR PIC S99 COMP-3.
36 003600 05 NO-OF-DEPENDENTS PIC X(26)
37 003700     VALUE "01000012000123001234002340".
38 003800 05 DEPENO REDEFINES NO-OF-DEPENDENTS PIC X OCCURS 26 TIMES.
39 003900 01 WORKRECORD.
40 004000 05 NAMES-FIELD PICTURE X.
41 004100 05 FILLER PICTURE X VALUE IS SPACE.
42 004200 05 RECORDNO PICTURE S999.
43 004300 05 FILLER PICTURE X VALUE IS SPACE.
44 004400 05 LOCATION PICTURE AAA VALUE IS "NYC".
45 004500 05 FILLER PICTURE X VALUE IS SPACE.
46 004600 05 DEPENDENTS PICTURE XX.
47 004700 05 FILLER PICTURE X(7) VALUE IS SPACES.
48 004800 01 NODEPEND PICTURE 99 COMP-3.
49 004900 01 ZONE-DATA PICTURE 99.
50 005000 PROCEDURE DIVISION.
005100*****
005200** THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO BE CREATED**
005300** AND INITIALIZES COUNTERS. **
005400*****
005500 STEP-1.
51 005600 OPEN OUTPUT FILE-1.

```

Figure 10-6 (Part 1 of 8). COBOL Formatted Dump


```

                                COBOL SOURCE LISTING          TESTPRT
STMT SEGNBR -A 1 B. . . . . 2 . . . . . 3 . . . . . 4 . . . . . 5 . . . . . 6 . . . . . 7 .IDENTFCN S COPYNAME  CHG/DATE

52 005700 MOVE ZERO TO KOUNT NUMBR NODEPEND.
005800*****
005900** THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE ***
006000** CONTAINED IN THE FILE, WRITES THEM ON DISK, AND DISPLAYS ***
006100** THEM. ***
006200*****
006300 STEP-2.
53 006400 ADD 1 TO KOUNT, ADD 1 TO NUMBR.
55 006500 MOVE ALPHA (KOUNT) TO NAMES-FIELD.
56 006600 MOVE DEPEND (KOUNT) TO DEPENDENTS.
57 006700 MOVE NUMBR TO RECORDNO.
006800 STEP-3.
58 006900 WRITE RECORD1 FROM WORKRECORD.
007000 STEP-4.
59 007100 PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
007200*****
007300** THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS ***
007400** INPUT. ***
007500*****
007600 STEP-5.
60 007700 CLOSE FILE-1.
61 007800 OPEN INPUT FILE-2.
007900*****
008000** THE FOLLOWING READS BACK THE FILE AND SINGLES OUT EMPLOYEES***
008100** WITH NO DEPENDENTS. ***
008200*****
008300 STEP-6.
62 008400 READ FILE-2 RECORD INTO WORKRECORD
63 008500 AT END GO TO STEP-6.
008600 STEP-7.
64 008700 IF DEPENDENTS IS EQUAL TO "0"
65 008800 MOVE "2" TO DEPENDENTS
66 008900 ADD 1 TO NODEPEND.
67 009000 GO TO STEP-6.
009100 STEP-8.
68 009200 CLOSE FILE-2.
69 009300 DISPLAY "EMPLOYEES WITH NO DEPENDENTS " NODEPEND.
70 009400 ADD 1 TO ZONE-DATA.
71 009500 STOP RUN.

***** END OF SOURCE *****

```

Figure 10-6 (Part 2 of 8). COBOL Formatted Dump

```

A MCH1202 EXCEPTION IN PROGRAM TESTPRT.PUBS AT MI INSTRUCTION NUMBER 0060 COBOL STATEMENT NUMBER 70.

B LAST I-D OPERATION WAS AT STATEMENT 68.

C CBE7903-INFORMATION PERTAINING TO FILE FILE-2.
CBE7904-FILE IS NOT OPEN.
CBE7906-LAST SUCCESSFUL I-D OPERATION FOR FILE WAS CLOSE.
CBE7907-LAST FILE STATUS FOR FILE WAS 00.

CBE7903-INFORMATION PERTAINING TO FILE FILE-1.
CBE7904-FILE IS NOT OPEN.
CBE7906-LAST SUCCESSFUL I-D OPERATION FOR FILE WAS CLOSE.
CBE7907-LAST FILE STATUS FOR FILE WAS 00.

```

Figure 10-6 (Part 3 of 8). COBOL Formatted Dump

NAME	OFFSET	ATTRIBUTES	VALUE
DB-FORMAT-NAME	0009A4	CHAR(10)	' * *'
DEPEND	000972	CHAR(1)	DIMENSION(26)
	000989	(3-26)	'0'
DEPENDENTS	000996	CHAR(2)	'Z'
FIELD		NOT ADDRESSABLE	
FILE-1		NOT ADDRESSABLE	
FILE-2		NOT ADDRESSABLE	
FILLER	000954	CHAR(56)	' ?ABCDEFGHIJKLMNPQRSTUWXYZ ?01000012000123001234002340'
	000954	VALUE IN HEX	'026FC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6D7D8D9E2E3E4E5E6E7E8E9026FF0F1F0F0F0F1F2F0F0*X'
	00097C	+41	'F0F1F2F3F0F0F1F2F3F4F0F0F2F3F4F0*X'
FILLER	00098D	CHAR(1)	' *'
FILLER	000998	CHAR(7)	' *'
FILLER	000995	CHAR(1)	' *'
FILLER	000991	CHAR(1)	' *'
KOUNT	000954	PACKED(2,0)	26
LOCATION	000992	CHAR(3)	'NYC'
NAMES-FIELD	00098C	CHAR(1)	'Z'
NO-OF-DEPENDENTS	000972	CHAR(26)	'01000012000123001234002340'
NODEPEND	0009A0	PACKED(2,0)	13
NULLCL	000901	CHAR(1)	' *'
NUMR	000970	PACKED(2,0)	26
RECORDNO	00098E	ZONED(3,0)	26
RECORD1		NOT ADDRESSABLE	
RECORD2		NOT ADDRESSABLE	
WORKRECORD	00098C	CHAR(19)	'Z 026 NYC Z *'
ZONE-DATA	0009A2	ZONED(2,0)	'**INVALID DATA *0000*X'

STATIC STORAGE FOR PROGRAM TESTPRT.PUBS BEGINS AT OFFSET 0000A0 IN THE PROGRAM STATIC STORAGE AREA (PSSA)
 AUTOMATIC STORAGE FOR PROGRAM TESTPRT.PUBS BEGINS AT OFFSET 0009A0 IN THE PROGRAM AUTOMATIC STORAGE AREA (PASA)

Figure 10-6 (Part 8 of 8). COBOL Formatted Dump



Chapter 11. COBOL Problem Determination

If a problem occurs while you are using COBOL, the cause of the problem may not be obvious. An error in your application, in system operation, or in the COBOL compiler could have caused the problem. The problem determination procedure in this chapter can help you solve or circumvent the problem. If you need to call for service, this chapter refers you to the *Problem Determination Guide*.

How to Use This Procedure

This procedure is arranged in a sequence of questions that you can answer with a YES or NO. Based on your answer, you are directed to another question or to a recommendation for action.

Start at the beginning of the procedure and follow the question-and-answer sequence, answering each question to which you are directed based on your previous answer. If the problem is a condition that requires more detailed procedures, you are referred to those procedures.

Identifying COBOL Problems

When a COBOL problem occurs, you can use the following series of questions to pinpoint its possible cause:

Have changes been made to the user program since the last time it compiled or ran successfully?

NO YES

Read on, but consider what has been changed. For example, have operating procedures changed, are new device files being used, or have program changes been applied recently? A good starting point for problem determination can be a changed item.

Are you using the current release of the COBOL compiler? The release number is printed on the first line of the source listing.

YES NO

Install the current release of the compiler and the program changes that apply to the release, and recompile the program. Refer to the

Guide to Program Product Installation and Device Configuration for an explanation of installing the compiler.

Have all IBM-supplied program changes you have received that apply to the current release of the compiler been installed?

YES NO

Install the program changes you have received that have not yet been applied, and recompile the program. Refer to the *Operator's Guide* for an explanation of installing program changes.

Are you using the current release of the CPF?

YES NO

Install the current release of the CPF and any program changes you have received that apply to the CPF release. Refer to the *Guide to Program Product Installation and Device Configuration* for an explanation of installing the CPF.

Have all IBM-supplied program changes you have received that apply to the current release of the CPF been installed?

YES NO

Install the program changes you have received that have not yet been applied. Refer to the *System/38 Operator's Guide* for an explanation of installing program changes.

Have any non-IBM-supplied modifications been made to the compiler or to the CPF?

NO YES

If the compiler has been changed, install its current release and program changes, and recompile the program. If the CPF has been changed, install its current release and program changes. Refer to the *Guide to Program Product Installation and Device Configuration* for an explanation of installing the compiler and the CPF.

Did the COBOL compiler have an exception?

NO YES

An exception is an error indicating an abnormal compiler termination. The exception is displayed at the work station or printed in the job log for the job that requested the compile. Exception data is printed and can be used to investigate the problem. For example, the name of the compiler phase executing and the number of the COBOL source statement being processed at the time of the exception are printed. You can refer to the source statement in the user program and try to modify the code to circumvent the problem. Retry the compilation and specify the DUMP parameter on the Create COBOL Program

command. Whether or not you successfully circumvent the problem, you should report it to your service representative.

Refer to “Calling for Help” later in this chapter.

Did the COBOL compiler loop or wait while compiling a user program?

NO YES

A loop or a wait is a seemingly never-ending compilation for which neither output nor error messages are produced.

For either condition you can (1) enter service mode for the job, (2) request a dump of the job, and (3) cancel the job. Refer to the *IBM System/38 Operator's Guide*, SC21-7735, for descriptions of entering service mode, requesting a dump, and cancelling a job.

Use the invocation stack layout at the beginning of the dump to investigate the problem. If you cannot circumvent or solve the problem, refer to “Calling for Help” later in this chapter.

Did the COBOL compiler produce incorrect output?

NO YES

One of the following conditions can indicate incorrect compiler output:

- An IRP syntax error, which causes the compiler to end abnormally and an error message to be issued to the compiler requester. IRP, which is an intermediate representation of a program that is produced by the compiler, is described in Appendix A.
- An unexpected result produced by a user program during its execution.

IRP Syntax Error: If your program caused an IRP syntax error, the IRP will be listed and you can trace the incorrect IRP statement back to the COBOL source statement(s) that produced it. Try to modify the COBOL source program to circumvent the error and then recompile the program. Appendix A provides information about relating IRP to COBOL source statements and using compiler debugging options such as *LIST.

Even if you circumvent the problem, you should also report it to your service representative. If you cannot circumvent the problem, refer to “Calling for Help” later in this chapter.

Unexpected Results: If a user program produces unexpected results, you can use CPF debugging functions such as traces and breakpoints, or you can use COBOL debugging feature USE FOR DEBUGGING statement to isolate problems within a user program. Refer to Chapter 6 for a description of COBOL debugging features and to Chapter 10 for an overview of CPF debugging functions.

If after investigating the problem you suspect a compiler error, recompile the program and specify *LIST for the GENOPT parameter on the Create COBOL Program command. The IRP will be listed, and you can trace the incorrect IRP statement back to the COBOL source statement(s) that produced it. Try to modify the COBOL source program to circumvent the error and then recompile the program. Appendix A provides information about relating IRP to COBOL source statements and using compiler debugging options such as *LIST.

If you cannot circumvent the problem, refer to “Calling for Help” later in this chapter.

Did the COBOL user program have an exception/error?

NO YES

Two types of exception/error can occur: program and file. Examples of program exception/errors are division by zero, use of an invalid index, and use of uninitialized data items in an arithmetic operation. Examples of file exception/errors are undefined record types and device errors.

You can begin investigating the problem at the source statement indicated by the message. Use CPF debugging functions or COBOL Debugging features to pinpoint the problem. Chapter 6 describes COBOL Debugging features, and Chapter 10 provides an overview of CPF debugging functions.

If you cannot solve or circumvent the problem, refer to “Calling for Help” later in this chapter.

Did the COBOL user program loop or wait during its execution or produce incorrect output?

NO YES

A loop is a sequence of instructions that is executed repeatedly, and a wait is a situation in which neither output nor error messages are produced.

For either condition you can (1) enter service mode for the job, (2) request a dump of the job, and (3) cancel the job. Refer to the *IBM System/38 Operator's Guide*, SC21-7735, for descriptions of entering service mode, requesting a dump, and canceling a job.

Use the invocation stack at the beginning of the dump to investigate the problem. Also check for program logic errors; if the program appears to be coded correctly, you can use CPF debugging functions or COBOL debugging features to pinpoint the problem. For example, you can trace the execution of certain statements and display the contents of fields at stopping points in the program. Refer to Chapter

6 for a description of COBOL debugging features and to Chapter 10 for an overview of CPF debugging functions.

When you isolate the problem to one or more source statements, you can list the IRP by recompiling and specifying *LIST for the GENOPT parameter on the Create COBOL Program command. If the IRP does not appear to be an accurate representation of the COBOL source statement, the compiler might be generating incorrect output. Appendix A provides IRP information for further investigating this type of problem.

If you cannot solve or circumvent the problem, refer to “Calling for Help” later in this chapter.

Refer to “Calling for Help” later in this chapter.

Calling for Help

If you require additional assistance, do the following:

1. Cancel the failing job and print the job log, if you have not already done so. Do this by signing off your work station and choosing *LIST for the OUTPUT parameter. For example:

```
SIGNOFF OUTPUT(*LIST)
```

Call your system operator to verify that the job log was printed.

2. Examine the job log, and any other available information on your job, to determine why your problem occurred. If you still require additional assistance, go to the next step.
3. Before calling for service, have your system operator go to the *System/38 Problem Determination Guide*.



Appendix A. COBOL Compiler Service Information

This appendix is provided for COBOL service personnel to use when investigating COBOL problems and provides the following information:

- Compiler overview
- Compiler debugging options
- IRP layout.

COBOL programmers can also use this information to investigate COBOL problems on their own before, or instead of, calling for service.

For more detailed information, refer to the microfiche for phase QCBROOT.

Compiler Overview

This section provides the following compiler information:

- How the compiler works
- Compiler phase descriptions
- Execution time subroutines
- Major compiler data areas descriptions
- Compiler error message organization.

This section provides an internal view of the compiler. If you need an external view to investigate a COBOL problem, refer to Chapter 8, which describes entering a COBOL program into the system, compiling the program, and using the listings that the compiler produces.

Figure A-1 summarizes how a COBOL source program is compiled into an executable (encapsulated) program.

Intermediate text, which is output from step 1 in Figure A-1, is a representation of COBOL source statements that is created by compiler phases and exists only during their execution. This text can be dynamically listed through the ITDUMP parameter of the CRTCLPGM command. See “Compiler Debugging Options” later in this appendix for explanations of these parameters and examples of intermediate text.

When compilation ends, intermediate text has been processed and converted to appropriate IRP (intermediate representation of a program). IRP, which is output from step 2 in Figure A-1, can be listed through an *LIST value for the GENOPT parameter on the CRTCLPGM command. See "Compiler Debugging Options" later in this appendix for explanations of these parameters and examples of IRP statements.

A program template is output from step 3 in Figure A-1. A template is the final form of a program before it is converted to an executable program, which is called an encapsulated program. A template can be listed at the end of a compilation through an *DUMP value for the GENOPT parameter on the CRTCLPGM command. See "Compiler Debugging Options" for an explanation of this parameter and an example of a program template listing.

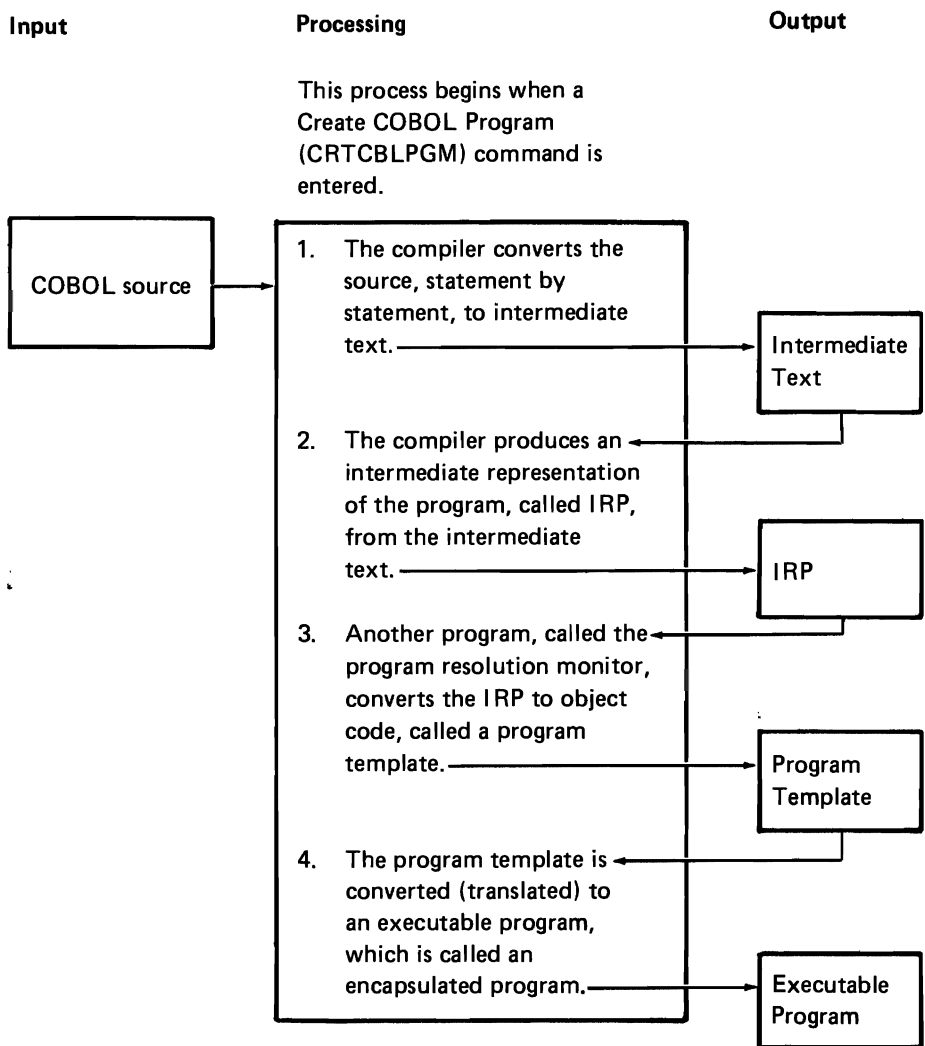


Figure A-1. Overview of the COBOL Compiler

Compiler Phases

The compiler consists of the phases listed in Figure A-2. These phases are shown in the order in which they are executed.

The first compiler phase is named QCBCL. The names of all phases that follow QCBCL begin with QCB and end with the function of the phase.

Execution-Time Subroutines

Figure A-3 lists the execution-time subroutines that are used by the compiler.

Major Compiler Data Areas

The major compiler data areas are a common area (XCBCOM), the dictionary (DHPACKET), and intermediate text (ITXT). Detailed information about these data areas can be found in the microfiche listing for phase QCBGEN.

Compiler Error Message Organization

Compiler error messages are organized as follows:

Error Messages	Description
CBL0000 to CBL0999	Messages with severity less than 30
CBL1000 to CBL1999	Messages with severity greater than 29
CBL8000 to CBL8999	FIPS messages

The *Messages Guide: COBOL* describes all compiler messages.

Phase Name	Phase Description
QCBABL	Command interface that receives control when the CRTCLPGM command is entered, and passes the command parameter list to QCBROOT.
QCBROOT	Root phase that controls the calling of all other compiler phases. This phase also sets the options from the command and the PROCESS statement.
QCBDATA	Phase that processes the first three COBOL divisions. The dictionary is built and those phrases that have dependencies on phrases that have not yet been processed are written to I-text to be processed by the next phase.
QCBENVIR	Phase that processes the I-text from the first three divisions.
QCBPROC	Phase that processes the Procedure Division statements. The COBOL statements are broken down into basic operations that are written to I-text.
QCBPARA	Phase that resolves procedure-names. All symbolic procedure-names in the I-text are resolved to dictionary entries.
QCBGEN	Phase that generates the IRP. The dictionary is scanned to produce the declarations and then the I-text is read to generate the IRP code for the Procedure Division statements.
QCBLIST	Phase that produces the listings. The verb sum list, Data Division map, FIPS flagging, cross-reference, and COBOL messages listing.
QCB OBJ	Phase that calls the PRM to create the program.

Figure A-2. Compiler Phases

Subroutine Name	Subroutine Description	Called: Unconditionally (U) Conditionally (C)
QCRMAIN	Initializing the COBOL program	U
QCRADRTN	Run-time routine for ACCEPT and DISPLAY	C
QCRANEDT	Run-time routine for doing alphanumeric editing	C
QCRCLEAR	Run-time routine to clear a file	C
QCREXHAN	Run-time routine to provide a symbolic dump	C
QCRFPWR	Run-time routine for noninteger exponentiation	C
QCRGDDM	Run-time routine for CALL "GDDM"	C
QCRIPWR	Run-time routine for integer exponentiation	C
QCRNAMCK	Run-time routine to verify that a name is a valid COBOL name and convert it if necessary	C
QCRSORT	Run-time routine for SORT/MERGE	C
QCRSTR	Run-time routine for STRING	C
QCRUNST	Run-time routine for UNSTRING	C

Figure A-3. Execution-Time Subroutines

Compiler Debugging Options

Figure A-4 shows parameters in the CRTCLPGM command that can be used to debug COBOL problems. This section explains each of these parameters. For information on the other parameters, see "COBOL Command Statement" in Chapter 8. For examples of debugging information that can be requested by these parameters, see "Examples of Using Compiler Debugging Options", later in this appendix.

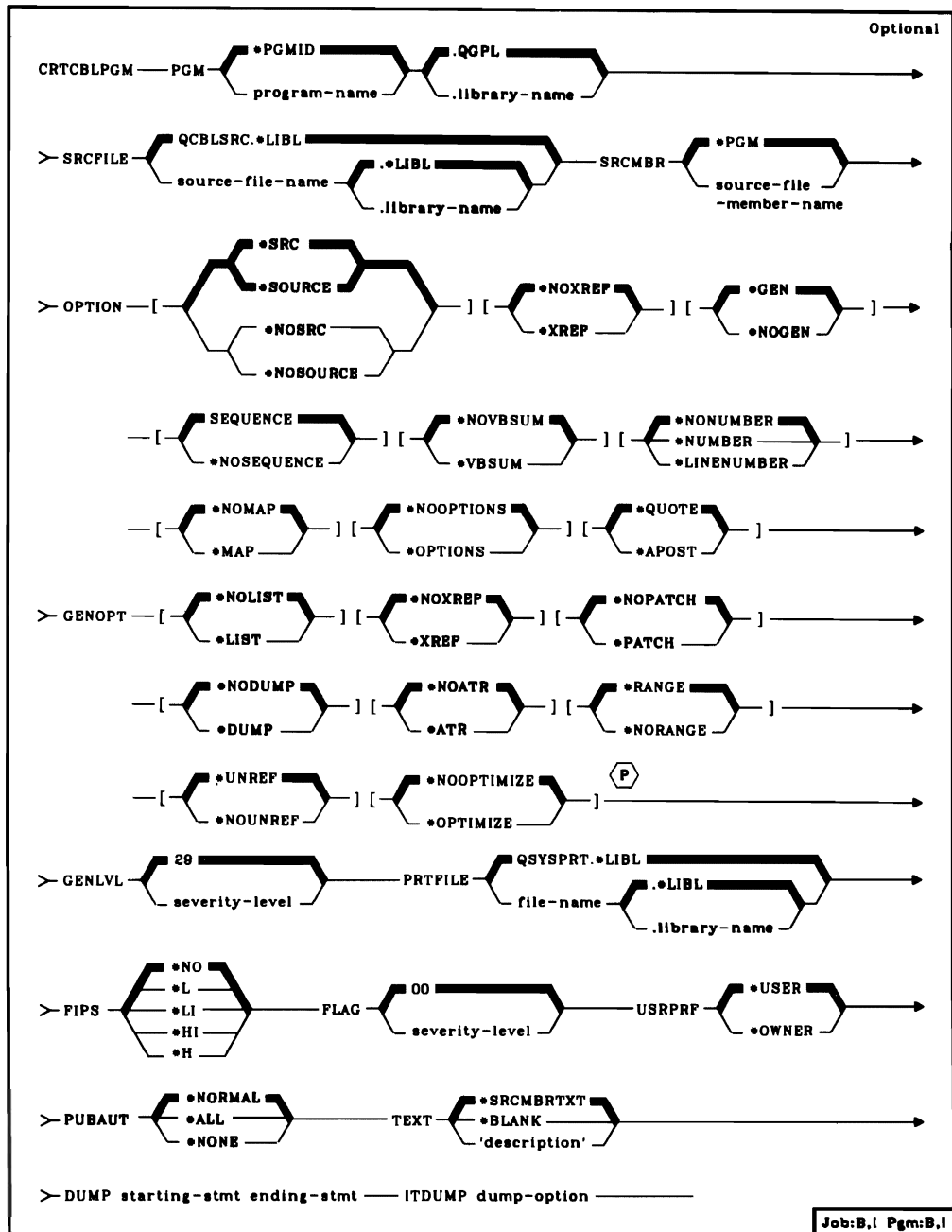


Figure A-4. Debugging Parameters

DUMP Parameter

The DUMP parameter produces a dictionary dump and also dumps compiler flow information and data for the statements specified.

This parameter is intended for use by IBM service personnel only.

ITDUMP Parameter

The ITDUMP parameter produces a dump of the I-text. The parameter is a number from 1 to 31 that controls when the I-text is dumped.

- 1 Dump the I-text after phase QCBPARA runs.
- 2 Dump the I-text before phase QCBPARA runs.
- 4 Dump the I-text as phase QCBGEN reads the I-text.
- 8 Dump the I-text as phase QCBGEN reads the I-text for the statements specified by the DUMP parameter.
- 16 Dump the I-text in phase QCBLIST.

Note: Any combination can be specified by adding the values together for the lists desired.

This parameter is intended for use by IBM service personnel only.

Examples of Using Compiler Debugging Options

Figure A-5 shows examples of debugging information that can be requested by compiler debugging options on the CRTCLPGM command. The compiler listing in Figure A-5 was printed for a CRTCLPGM command that specified debugging parameters as follows:

```
GENOPT(*LIST *DUMP) DUMP(53 54) ITDUMP(4)
```

The DUMP(53 54) parameter causes a dump of tokens, semantic exits, all dictionary activity, and all I-text written for statements 53 through 54. (See **A** in Figure A-5.)

Whenever the DUMP parameter is specified, a dump of the dictionary is printed at the end. (See **C** in Figure A-5.)

The ITDUMP(4) parameter causes printing of the intermediate text as it is processed by phase QCBGEN. The name of the routine that processes the intermediate text is listed in front of each record. (See **B** in Figure A-5.)

The *LIST value for the GENOPT parameter causes printing of IRP and machine instructions when compilation ends. (See **H** in Figure A-5.) The headings in this IRP listing indicate the following information:

- *SEQ*: A sequential numbering of the IRP statements. Error messages such as IRP syntax errors issued by the program resolution monitor use this number to refer to the IRP statements in error.

- **INST:** A sequential numbering of the machine instructions generated from the IRP statements. Not all IRP statements cause machine instructions to be generated. The instruction number can be used as a breakpoint for CPF debugging functions. Refer to Chapter 10 or the *CPF Programmer's Guide* for further information about breakpoints.
- **OFFSET:** Displacement of the machine instruction into the instruction portion of the program template.
- **GENERATED CODE:** Machine instructions that have been generated from IRP statements.
- **GENERATED OUTPUT:** IRP statements.
- **BREAK:** Breakpoints in the IRP that can be used for stopping points in CPF debugging functions. If the breakpoint is a number, it indicates the COBOL source statement from which the IRP statement was generated.

The following chart lists breakpoint names and the COBOL functions that they handle. These breakpoint names can appear in error messages. They help the user relate the statement number (field) to the function being performed.

Breakpoint Name	COBOL Function
.CALEXCP	ON OVERFLOW exception monitor
.CLOSE	Close file
.CNLEXCP	CANCEL exception monitor
.CPF5004	Page overflow exception monitor
.CTRL	Control-Area processing
.DEBUGBLD	Debug execution table build routine
.DEBUGOUT	Debug table maintenance code
.DELETE	Delete routine
.EXCKRD	Called before doing a delete operation or a rewrite operation
.EXFSGN	Generic exception monitor
.EXFS10	End of file exception monitor
.EXFS11	No data available for file exception monitor
.EXFS12	No modified subfile record found exception monitor
.EXFS22	Duplicate key exception monitor

Breakpoint Name	COBOL Function
.EXFS23	Record not found exception monitor
.EXFS24	Boundary violation exception monitor
.EXFS30	Permanent error or hardware error exception monitor
.EXFS34	Permanent error or boundary violation exception monitor
.EXFS9D	A record is locked
.EXFS9G	Input buffer too small exception monitor
.EXFS9I	Write operation failed exception monitor
.EXFS9K	Invalid format name exception monitor
.EXFS9M	Last record written to subfile exception monitor
.EXFS9N	Recoverable I-O error exception monitor
.EXFS9P	Commitment control exception monitor
.EXFS91	Invalid or unauthorized access exception monitor
.EXFS92	Logic error exception monitor
.EXINVT	Device was acquired but not invited
.EXISLD	Called before doing a sequential PUT to an indexed file
.EXNOF	File or member not found exception monitor
.EXRWRT	Called before doing a rewrite operation to an indexed file with sequential access
.EXSB23	Subfile record not found exception monitor
.EXSB24	Subfile boundary violation exception monitor
.FCEXCP	Function check exception monitor
.FEOV	Force end of volume
.GET	GET
.GETKEY	Reads the record by the key value
.GETREL	GET relative
.INIT	Initialization code and declares
.INIT002	Initialize return pointer
.LSKA	Linage control: before advancing page

Breakpoint Name	COBOL Function
.LSKB	Linage control: after advancing page
.LSPA	Linage control: before advancing n lines
.LSPB	Linage control: after advancing n lines
.LSTRT	Linage initialization
.NOALTR	Unaltered GOTO
.OPEN	Open file
.PAGING	Paging routine
.PRINIT	Printer initialization
.PRPUT	Printer PUT
.PSPA	Print control: before advancing
.PSPB	Print control: after advancing
.PUTALL	PUT
.PUTGET	PUTGET
.PUTKEY	Indexed keyed PUT
.REWRIT	Rewrite
.SIZEXCP	Internal size error exception monitor
.STARTR	Relative I/O start
.STPEXCP	STOP RUN exception monitor
.USETEST	Save the CFIB address

See Chapter 10 or the *CPF Programmer's Guide* for further information on breakpoints.

The *DUMP value for the GENOPT parameter causes printing of the program template when compilation ends. (See **L** in Figure A-5.)

IRP Layout

The COBOL compiler generates code that has the following sections:

- Declaration of MGT (see **D** in Figure A-5). The Module Global Table (MGT) defines a common area for the module. This table is used to pass information to execution-time subroutines.

- Declaration of PGT (see **E** in Figure A-5). The Program Global Table (PGT) is a communication area for the entire COBOL run unit (there is only one PGT for the run unit).
- Declarations for Environment and Data Division (see **F** in Figure A-5). The declarations are generated in the same order as they appear in the COBOL source. The HLL keyword gives the source COBOL name.
- Procedure Division Code (see **J** in Figure A-5). This lists the generated code for the Procedure Division statements. The BRK statement identifies the COBOL statement number, and the comment indicates the COBOL verb.
- Paragraph Initialization Table (see **G** in Figure A-5). This table contains information about paragraphs that are performed.
- Compiler-Generated Subroutines (see **K** in Figure A-5). Common subroutines are included as needed for each program.

Some of the labels that can appear in the IRP for a COBOL program are:

.INIT	Label on the first executable MI instruction in the generated program.
.Lnnnnnn	Compiler-generated labels where nnnnnn is a six digit number.
.PARnnnn	Generated for each paragraph-name in the Procedure Division, where nnnn is the paragraph number. The paragraphs are defined in the Procedure Division.
.SECnnnn	Generated for each section-name in the Procedure Division, where nnnn is the section number. The sections are numbered sequentially starting with 1 as they are defined in the Procedure Division.
.RETURN	Label on the code for program termination.

```

                                COBOL SOURCE LISTING
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S  COPYNAME  CHG/DATE

1 000100 PROCESS OPTIONS MAP
      COBOL COMPILER OPTIONS IN EFFECT

      OPTIONS
      SOURCE
      NOXREF
      MAP
      NOVBSUM
      NONNUMBER
      SEQUENCE
      GEN
      GENMLL(29)
      FLAG( 0)
      FIPS(NO)
      QUOTE
      DUMP( 53, 54)
      ITDUMP( 4)

      COBOL GENERATION OPTIONS IN EFFECT

      LIST
      UNREF
      RANGE
      NDATR
      NOXREF
      DUMP
      NOPATCH

```

```

                                COBOL SOURCE LISTING
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S  COPYNAME  CHG/DATE

2 000200 IDENTIFICATION DIVISION.
3 000300 PROGRAM-ID. TESTPRT.
4 000400 AUTHOR. PROGRAMMER NAME.
5 000500 INSTALLATION. ROCHESTER LABORATORY.
6 000600 DATE-WRITTEN. JULY 15, 1980.
7 000700 DATE-COMPILED. 11/06/80 18:46:00 .
8 000800 ENVIRONMENT DIVISION.
9 000900 CONFIGURATION SECTION.
10 001000 SOURCE-COMPUTER. IBM-538.
11 001100 OBJECT-COMPUTER. IBM-538.
12 001200 INPUT-OUTPUT SECTION.
13 001300 FILE-CONTROL.
14 001400 SELECT FILE-1 ASSIGN TO DATABASE-MASTER.
15 001500 SELECT FILE-2 ASSIGN TO DATABASE-MASTER.
16 001600 DATA DIVISION.
17 001700 FILE SECTION.
18 001800 FD FILE-1
19 001900 LABEL RECORDS ARE STANDARD
20 002000 RECORD CONTAINS 20 CHARACTERS
21 002100 DATA RECORD IS RECORD1.
22 002200 01 RECORD1.
23 002300 02 FIELD1 PICTURE IS X(20).
24 002400 FD FILE-2
25 002500 LABEL RECORDS ARE STANDARD
26 002600 RECORD CONTAINS 20 CHARACTERS
27 002700 DATA RECORD IS RECORD2.
28 002800 01 RECORD2.
29 002900 02 FIELD2 PICTURE IS X(20).
30 003000 WORKING-STORAGE SECTION.
31 003100 01 FILLER.
32 003200 05 KOUNT PIC S99 COMP-3.
33 003300 05 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNPOQRSTUVWXYZ".
34 003400 05 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
35 003500 05 NUMBR PIC S99 COMP-3.
36 003600 05 NO-OF-DEPENDENTS PIC X(26)
37 003700 05 VALUE "01000012000123001234002340".
38 003800 05 DEPEND REDEFINES NO-OF-DEPENDENTS PIC X OCCURS 26 TIMES.
39 003900 01 WORKRECORD.
40 004000 05 NAMES-FIELD PICTURE X.
41 004100 05 FILLER PICTURE X VALUE IS SPACE.
42 004200 05 RECORDNO PICTURE S9999.
43 004300 05 FILLER PICTURE X VALUE IS SPACE.
44 004400 05 LOCATION PICTURE AAA VALUE IS "NYC".
45 004500 05 FILLER PICTURE X VALUE IS SPACE.
46 004600 05 DEPENDENTS PICTURE XX.
47 004700 05 FILLER PICTURE X(7) VALUE IS SPACES.
48 004800 01 NDEPEND PICTURE 99 COMP-3.
49 004900 01 ZONE-DATA PICTURE 99.

DATA DIVISION SEMANTIC EXIT COUNTS
COUNTS FOR EXITS 1 - 10 0 0 0 0 0 0 0 0 0
COUNTS FOR EXITS 11 - 20 3 0 0 0 22 3 0 0 0 1
COUNTS FOR EXITS 21 - 30 1 0 1 4 0 0 0 0 12 6
COUNTS FOR EXITS 31 - 40 0 0 0 0 0 0 0 0 22 2

```

Figure A-5 (Part 1 of 10). Examples of Compiler Debugging Information


```

                                COBOL SOURCE LISTING                                TESTPRT
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME  CHG/DATE

COUNTS FOR EXITS 41 - 50      0 0 2 16 0 0 2 7 2 16
COUNTS FOR EXITS 51 - 60      2 2 0 7 4 0 0 3 0 1
COUNTS FOR EXITS 61 - 70      0 0 0 0 0 0 2 0 0 2
COUNTS FOR EXITS 71 - 80      0 0 0 0 0 0 0 12 12 2
COUNTS FOR EXITS 81 - 90      0 3 4 6 2 2 6 1 2 0

TOTAL NUMBER OF EXITS TAKEN IN QCB0TA:      199

ENVIRONMENT ITEXT SEMANTIC EXIT COUNTS
COUNTS FOR EXITS 1 - 10      0 0 0 0 0 0 0 0 2 1
COUNTS FOR EXITS 11 - 20     1 4 0 2 0 0 0 0 2 2 0
COUNTS FOR EXITS 21 - 30     0 2 0 0 0 0 0 0 0 0
COUNTS FOR EXITS 31 - 40     0 0 0 0 0 0 2 2 2 0
COUNTS FOR EXITS 41 - 50     0 0 0 0 0 0 0 0 0 0
COUNTS FOR EXITS 51 - 60     0 0 4 0 0 0 4 2 0 0
COUNTS FOR EXITS 61 - 70     2 2 2 2 0 4 2 0 0 2
COUNTS FOR EXITS 71 - 80     23856-3856-3856-3956-3856-3856-3856-3856-3856-3856-
50 005000 PROCEDURE DIVISION.
005100*****
005200** THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO BE CREATED**
005300** AND INITIALIZES COUNTERS. **
005400*****
005500 STEP-1
51 005600 OPEN OUTPUT FILE-1.
52 005700 MOVE ZERO TO KOUNT NUMBR NODEPEND.
005800*****
005900** THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE **
006000** CONTAINED IN THE FILE, WRITES THEM ON DISK, AND DISPLAYS **
006100** THEM. **
006200*****
006300 STEP-2.
53 006400 ADD 1 TO KOUNT. ADD 1 TO NUMBR.
OCBPRSE: TOKEN IS- 101 NJMNR
00000000 000000000000000000000000003640F0F0F6F4F0F0000022
OCBPR: SEMANTIC - 21 101 5 NUMBR
OCBPRSE: TOKEN IS- 117 .
00000000 000000000000000000000000003640F0F0F6F4F0F0000042
OCBPR: SEMANTIC - 24 117 2 .

00005710 NUMBR
DHCODE = DATA-NAM DHTL = 82 DHTP = 11000000
DHPARENT = 00005580 DMLINE = 35 DHSQNR = 003500
DHDEBUG = 00000000 DHMASH = 00000000 DHTWIN = 00000000
DHTP = 11000000 00000000
DDINLEVL = 2 DDLEN = 2 DDFLAG = 00000000 00000000
DDSECT = WORK DDSSUB = 0 DDS = 01000001
DDC = PACKED DDADDR = 28 DDDT,OR = ( 2, 0)
DDXLEVL = 5 DDACKCHN = 00000000
DDOPOS = 0 DDREDEF=00000000
OCBPR: SEMANTIC - 82 101 5 NUMBR
OCBPR: SEMANTIC - 84 117 2 .
OCBPR: SEMANTIC - 116 117 2 .
OCBPR: SEMANTIC - 112 117 2 .
00003228: *-TUPLE ADD 2: 00 80 00 41 18 2 0000 0002 00005710 NUMBR

```

```

                                COBOL SOURCE LISTING                                TESTPRT
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME  CHG/DATE

                                00 01 00 40 21 1 0000 0001 F1404040
OCBPR: SEMANTIC - 115 117 2 .
OCBPR: SEMANTIC - 12 117 2 .
OCBPRSE: TOKEN IS- 29 MOVE
00003000 000000000000000000000000003640F0F0F6F5F0F000008
OCBPRU: SEMANTIC - 126 29 4 MOVE
OCBPR: SEMANTIC - 13 29 4 MOVE
OCBPR: SEMANTIC - 12 29 4 MOVE
00003250: STMT HDR 55 006500 000000 1- 29
OCBPRSE: TOKEN IS- 101 ALPHA
00000000 000000000000000000000000003740F0F0F6F5F0F000000E
OCBPR: SEMANTIC - 21 101 5 ALPHA
OCBPRSE: TOKEN IS- 113 (
00000000 000000000000000000000000003740F0F0F6F5F0F0000010
OCBPR: SEMANTIC - 24 113 1 (

00005680 ALPHA
DHCODE = DATA-NAM DHTL = 82 DHTP = 00000000
DHPARENT = 00005580 DMLINE = 34 DHSQNR = 003400
DHDEBUG = 00000000 DHMASH = 00000000 DHTWIN = 00000000
DHTP = 00000000 00000000
DDINLEVL = 2 DDLEN = 1 DDFLAG = 00010100 00000000
DDSECT = WORK DDSSUB = 1 DDS = 00000000
DDC = AN DDADDR = 2 DDLATR = 1
DDXLEVL = 5 DDACKCHN = 00000000
DDOPOS = 0 DDREDEF=00005634
OCBPR: SEMANTIC - 82 101 5 ALPHA
OCBPR: SEMANTIC - 36 113 1 (
OCBPR: SEMANTIC - 83 113 1 (
OCBPRSE: TOKEN IS- 101 KOUNT
00000000 000000000000000000000000003740F0F0F6F5F0F0000015
OCBPR: SEMANTIC - 21 101 5 KOUNT
OCBPRSE: TOKEN IS- 121 )
00000000 000000000000000000000000003740F0F0F6F5F0F0000016
OCBPR: SEMANTIC - 24 121 1 )

00005502 KOUNT
DHCODE = DATA-NAM DHTL = 82 DHTP = 11000000
DHPARENT = 00005580 DMLINE = 32 DHSQNR = 003200
DHDEBUG = 00000000 DHMASH = 00000000 DHTWIN = 00000000
DHTP = 11000000 00000000
DDINLEVL = 2 DDLEN = 2 DDFLAG = 00000000 00000000
DDSECT = WORK DDSSUB = 0 DDS = 01000001
DDC = PACKED DDADDR = 0 DDDT,OR = ( 2, 0)
DDXLEVL = 5 DDACKCHN = 00000000
DDOPOS = 0 DDREDEF=00000000
OCBPR: SEMANTIC - 85 121 1 )
OCBPR: SEMANTIC - 86 121 1 )
00003264: *-TUPLE SUBSCR 3: 00 80 00 00 0 1 0000 0000 00000000
00 80 00 00 11 1 0000 0001 00005680 ALPHA
00 80 00 41 0 22192 0000 0001 00005502 KOUNT

OCBPR: SEMANTIC - 37 121 1 )
OCBPR: SEMANTIC - 27 121 1 )
OCBPRSE: TOKEN IS- 834 TO

```

Figure A-5 (Part 2 of 10). Examples of Compiler Debugging Information


```

                                CDBOL SOURCE LISTING                TESTPRT
STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME  CHG/DATE

OCBGLBL : 000037E4: N-TUPLE LABEL          00 02 00 41 18      1 0000 0002 00000000
1:
00 20 00 00 0 0 0 0000 0000 00000000
OCBGEN : 000037FC: STMT HDR          67 009000 000000 1- 23
OCBGBRCH: 00003810: N-TUPLE BRANCH      1:
00 80 00 00 1 0 0007 0000 00007CC8 STEP-6
OCBGLBL : 00003834: N-TUPLE LABEL          1:
00 80 00 00 1 0 0000 0000 00007D40 STEP-8
OCBGEN : 0000384C: STMT HDR          68 009200 000000 1- 7
OCBGID : 00003860: N-TUPLE CLOSE        2:
00 80 00 00 3 0 0000 0000 0000543A FILE-2
00 20 00 00 0 0 0000 0000 00050000
OCBGEN : 00003888: STMT HDR          69 004300 000000 1- 13
OCBGDISP: 0000389C: N-TUPLE DISPLAY      3:
00 80 00 00 22 31 0000 001D 00007D7C "EMPLOYEES WITH NO DEPENDENTS "
00 80 00 00 18 2 0000 0002 000058BC NODEPEND
00 20 00 00 0 0 0000 0000 00000000
OCBGMOVE: 0001E2D4: N-TUPLE MOVE        2:
00 02 00 41 11 1 0000 001D 00000000
00 80 00 00 22 31 0000 001D 00007D7C "EMPLOYEES WITH NO DEPENDENTS "
OCBGMOVE: 0001E2D4: N-TUPLE MOVE        2:
00 02 00 41 16 1 0000 0002 00000000
00 80 00 00 18 2 0000 0002 000058BC NODEPEND
OCBGEN : 000038D4: STMT HDR          70 009400 000000 0 2
OCBGARTH: 000038E8: N-TUPLE ADD         2:
00 80 00 00 16 2 0000 0002 00005BEE ZONE-DATA
00 01 00 40 21 1 0000 0001 F1404040
OCBGMOVE: 0001B404: N-TUPLE MOVE        2:
00 02 00 41 16 1 0000 0002 00000000
00 80 00 00 16 2 0000 0002 00005BEE ZONE-DATA
OCBGMOVE: 0001B404: N-TUPLE MOVE        2:
00 02 00 41 16 1 0000 0002 00000000
00 80 00 00 16 2 0000 0002 00005BEE ZONE-DATA
OCBGEN : 00003910: STMT HDR          71 009500 000000 1- 43
OCBGSTOP: 00003924: N-TUPLE STOP        1:
00 20 00 00 0 0 0000 0000 00000000
OCBGLBL : 0000393C: N-TUPLE LABEL          1:
00 20 00 00 0 0 0000 0000 0000000F

* * * * * E N D O F S O U R C E * * * * *

```

```

                                CDBOL MESSAGES                    TESTPRT
STMT SEQNBR MSGID SEV TEXT

MESSAGE SUMMARY
TOTAL INFO(0-4) WARNING(5-19) ERROR(20-29) SEVERE(30-39) TERMINAL(40-99)
0 0 0 0 0 0

C DUMP OF DICTIONARY FOLLOWS

000052EC FILE-1
DHCODE = FILE-PKT DHTL = 168 DHTP = 10000000
DHPARENT = 00000000 DMLINE = 18 DHSONR = 001800
DHDEBUG = 00000000 DHMASH = 00000000 DHTWIN = 00000000
DHTP = 10000000 00000000
DFD = 1 DFORG = 1 DFACC = 1
DFSAME = 0 DFSAMRC = 0 DFASIGN = MASTER
DFSACHN = 00000000 DFSRCHN = 00000000 RERUN = 0
DFUSE = 00000000 DFCODSET = 00000000 DFFSTAT = 00000000
DFRELKY = 00000000 DFRECKY = 00000000 DFALTKY = 00000000
DFLBDY = 00000000 DFLTOP = 00000000 DFLBOT = 00000000
DFLFOOT = 00000000 DFMPRL = 20 DFMPRL = 20
DFRECL = 20 DFRECS = 20 DFILENO = 1
DFWKSTNO = 00000000 DFS = 00000010 00000000 00001000
DFASFN0 = 0 DFSRLN = 0 DFSTAT2 = 00000000
DFIOECP = 00001001 00101010
DFIINFO = 11001000 00000000

00005394 RECORD1
DHCODE = DATA-NAM DHTL = 84 DHTP = 11000000
DHPARENT = 000052EC DMLINE = 22 DHSONR = 002200
DHDEBUG = 00000000 DHMASH = 00000000 DHTWIN = 00000000
DHTP = 11000000 00000000
DDINLEVL = 1 DDLEN = 20 DDFLAG = 00000000 00000000
DDSECT = FILE DDSSUB = 0 DDS = 00000000
DDC = GROUP DDADDR = 0 DDLATR = 20
DDEKLEVL = 1 DDAKCHN = 00000000
DDQPOS = 0 DDREDEF = 00000000

000053E8 FIELDA
DHCODE = DATA-NAM DHTL = 82 DHTP = 00000000
DHPARENT = 00005394 DMLINE = 23 DHSONR = 002300
DHDEBUG = 00000000 DHMASH = 00000000 DHTWIN = 00005336
DHTP = 00000000 00000000
DDINLEVL = 2 DDLEN = 20 DDFLAG = 00000000 00000000
DDSECT = FILE DDSSUB = 0 DDS = 00000000
DDC = AN DDADDR = 0 DDLATR = 20
DDEKLEVL = 2 DDAKCHN = 00000000
DDQPOS = 0 DDREDEF = 00000000

0000543A FILE-2
DHCODE = FILE-PKT DHTL = 168 DHTP = 10000000
DHPARENT = 00000000 DMLINE = 24 DHSONR = 002400

```

Figure A-5 (Part 4 of 10). Examples of Compiler Debugging Information

STMT	SEQNBR	MSGID	SEV	TEXT	COBOL MESSAGES	TESTPRT
000057C		STEP-5				
		DHCDD = PROC-NM	DHTL =	60	DHTP =	00000000
		DHPARENT = 00005D48	DHL LINE =	60	DHSQNR =	007600
		DHDEBUB = 00000000	DHMHASH =	00005394	DHTWIN =	00000000
		DHTP = 00000000	00000000			
		DPFCFLAGS = 10000000	00000000		DPSMSTNR =	0
		DPPROCNR = 6	DPSEGNR =	0	DPAGQPNR =	0
		DPAGOSNR = 0	DPNXTNBR =	7	DPNXTSEG =	0
00007CCB		STEP-6				
		DHCDD = PROC-NM	DHTL =	60	DHTP =	10000000
		DHPARENT = 00005D48	DHL LINE =	62	DHSQNR =	008300
		DHDEBUB = 00000000	DHMHASH =	000054E2	DHTWIN =	00000000
		DHTP = 10000000	00000000			
		DPFCFLAGS = 10000010	00000000		DPSMSTNR =	0
		DPPROCNR = 7	DPSEGNR =	0	DPAGQPNR =	0
		DPAGOSNR = 0	DPNXTNBR =	8	DPNXTSEG =	0
00007D0A		STEP-7				
		DHCDD = PROC-NM	DHTL =	60	DHTP =	00000000
		DHPARENT = 00005D48	DHL LINE =	64	DHSQNR =	008600
		DHDEBUB = 00000000	DHMHASH =	00000000	DHTWIN =	00000000
		DHTP = 00000000	00000000			
		DPFCFLAGS = 10000000	00000000		DPSMSTNR =	0
		DPPROCNR = 8	DPSEGNR =	0	DPAGQPNR =	0
		DPAGOSNR = 0	DPNXTNBR =	9	DPNXTSEG =	0
00007D40		STEP-8				
		DHCDD = PROC-NM	DHTL =	60	DHTP =	10000000
		DHPARENT = 00005D48	DHL LINE =	68	DHSQNR =	009100
		DHDEBUB = 00000000	DHMHASH =	00000000	DHTWIN =	00000000
		DHTP = 10000000	00000000			
		DPFCFLAGS = 10000010	00000000		DPSMSTNR =	0
		DPPROCNR = 9	DPSEGNR =	0	DPAGQPNR =	0
		DPAGOSNR = 0	DPNXTNBR =	0	DPNXTSEG =	0
00007D7C		"EMPLOYEES WITH NO DEPENDENTS"				
		DHCDD = NONNUM-L	DHTL =	42		
		OCLLEN =		31		
		95 SOURCE RECORDS READ				
		0 COPY RECORDS READ				
		0 COPY MEMBERS PROCESSED				
		0 SEQUENCE ERRORS				
		45 DICTIONARY FINDS				
		51 DICTIONARY PUTS				
		0 WAS THE HIGHEST SEVERITY MESSAGE ISSUED				

SEQ	INST	OFFSET	GENERATED CODE	*... .. 1 2 3 4 5 6 7 8	BREAK
00001			TITLE *IBM 5/38 COBOL 5714C81 R02M00	IRP LISTING FOR TESTPRT	;
00002			ENTRY * EXT		;
00003			BRK *MGT *		;
00004			DCL DD *MGT CHAR(16) BDRY(16) INIT(*COBOL MGT 00.0*)		;
00005			DCL SPCPTR *MGTNEXT		;
00006			DCL SPCPTR *MGTFFIB		;
00007			DCL SPCPTR *MGTPTG		;
00008			DCL SPCPTR *MGTPTASA		;
00009			DCL SPCPTR *MGTSEPT		;
00010			DCL SPCPTR *MGTUPTR		;
00011			DCL SPSPTR *MGTTPGM		;
00012			DCL INSPTR *MGTTRST		;
00013			DCL DD *MGTCTR(20) BIN(4) INIT		;
			/*1 - SIZE EXCEPTIONS*/		;
00014			DCL DD *MGTNAME CHAR(10)		;
00015			DCL DD *MGTINVC BIN(2) INIT		;
00016			DCL DD *MGTEKCP CHAR(7) INIT		;
00017			DCL DD *MGTIND(32) CHAR(1) INIT(*00*)		;
00018			/*1-ON SIZE ERR IND*/		;
			/*2-STMT OSE IND*/		;
			/*3-OVFL EXCPT-CALL*/		;
			/*4-CNCL EXCPT-PRG ACTIVE*/		;
			/*5-FIRST TIME PERFORM FOR STATEMENT*/		;
			/*6-EXEC TIME DEBUG ON FOR PROGRAM*/		;
			/*7-PUT OUT SPECIAL DEBUG ITEMS*/		;
00019			DCL DD *MGTSZ CHAR(1) DEF(*MGTIND) POS(1)		;
00020			DCL DD *MGTSOZ CHAR(1) DEF(*MGTIND) POS(2)		;
00021			DCL DD *MGTOFL CHAR(1) DEF(*MGTIND) POS(3)		;
00022			DCL DD *MGTNCNCL CHAR(1) DEF(*MGTIND) POS(4)		;
00023			DCL DD *MGTTPM CHAR(1) DEF(*MGTIND) POS(5)		;
00024			DCL DD *MGTDBUG CHAR(1) DEF(*MGTIND) POS(6)		;
00025			DCL DD *MGTSPCD CHAR(1) DEF(*MGTIND) POS(7)		;
00026			DCL DD *MGTSM CHAR(1) INIT(*00*)		;
00027			/*BIT 1 - FIRST TIME SWITCH*/		;
00028			DCL DD *MGTTYPE CHAR(1)		;
00029			DCL DD *MGTFUNC BIN(2) INIT		;
00030			/*1 - STOP RUN*/		;
			/*2 - INITIAL CALL*/		;
			/*3 - CONVERT FROM PACKED 31 TO BINARY 8*/		;
			/*4 - CONVERT FROM BINARY 8 TO PACKED 31*/		;
			/*5 - REFRESH PROC PTRS FOR SEGMENTS*/		;
00031			DCL DD *MGTINB CHAR(8)		;
00032			/*BINARY 8 - CONVERSION*/		;

Figure A-5 (Part 5 of 10). Examples of Compiler Debugging Information

```

IBM S/38 COBOL S714C91      IRP LISTING FOR TESTPRT
SEQ  INST OFFSET  GENERATED CODE      *... 1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 ... 8 BREAK
00033          DCL DD .MGTPACK PKD(31,0)          /*PACKED 31 - CONVERSION*/ ; .MGT
00034          DCL DD .MGTSEG BIN(2)           /*TARGET SEGMENT #*/ ; .MGT
00035          DCL DD .MGTLVL BIN(2)           /*PARAM LEVEL*/ ; .MGT
00036          DCL DD * CHAR(13)              /*FOR ALGNMNT*/ ; .MGT
00037          DCL SPCPTR .MGTPCS             /*-> PGM CBL SEQ TBL*/ ; .MGT
00038          DCL SPCPTR .MGTPTR INIT(.PT)    /*-> PARAGRAPH TABLE*/ ; .MGT
00039          DCL SYSPTR .MGTCPCN           /*CANCELED PGM*/ ; .MGT
00040          DCL INSPTR .IORTN              /*RETURN POINT FROM I/O RT ; .MGT
          NS*/ ; .MGT
00041          DCL SPCPTR .FIBPTR            /*CURRENT FIB PTR*/ ; .MGT
00042          DCL INSPTR .MGTMSSG          /*->COBOL STMT*/ ; .MGT
00043          DCL SPCPTR .MGFMST           /*SUBSTITUTION TEXT*/ ; .MGT
00044          DCL SPCPTR .MGFMST           /*REPLY AREA*/ ; .MGT
00045          DCL DD .MGTMSSG BIN(2)         /*RELATIVE INVOCATION NBR*/ ; .MGT
          ; .MGT
00046          DCL DD .MGTMSSG CHAR(7)      /*MSG ID*/ ; .MGT
00047          DCL DD * CHAR(7)             /*UNUSED*/ ; .MGT
00048          DCL SPCPTR .MGTPARM          /*->ADDTNL PARM*/ ; .MGT
          ; .MGT
          /*END OF MGT*/ ; .MGT
00049          ; .MGT
00050          DCL CON .MGTCBT CHAR(1) INIT('B') /*BATCH JOB*/ ; .MGT
00051          DCL CON .MGTCIN CHAR(1) INIT('I') /*INTERACTIVE JOB*/ ; .MGT
00052          DCL SYSPTR .WCBSEPT(32767) BAS(.MGTSEPT) ; .MGT

00053          BRK *.WCBDTA*                /*WCB USER DATA AREA*/ ; .WCBDTA
00054          DCL SPC .WCBDTA BAS(.MGTUPTR) ; .WCBDTA
00055          DCL DD .WCBURC BIN(2) DIR      /*L/U RETURN CODE*/ ; .WCBDTA
          ; .WCBDTA
          /*0 - NORMAL*/ ; .WCBDTA
          /*1 - LR NOT SET*/ ; .WCBDTA
          /*2 - ERROR*/ ; .WCBDTA
          /*3 - HALT SET*/ ; .WCBDTA
          ; .WCBDTA
00056          DCL DD .WCBPINF BIN(2) DIR   /*PRODUCT INFO*/ ; .WCBDTA
00057          DCL DD * CHAR(10) DIR        ; .WCBDTA
00058          DCL DD .WCBURC CHAR(2) DIR   /*USER RETURN CODE AREA*/ ; .WCBDTA
00059          DCL DD .WCBM5 CHAR(1) DIR    /*CANCEL SW*/ ; .WCBDTA
00060          DCL DD .WCBJDT CHAR(7) DIR   /*JOB DATE GUARD,YYMMDD*/ ; .WCBDTA
00061          DCL DD .WCBSWC CHAR(8) DIR   /*UPSI SWITCHES*/ ; .WCBDTA
00062          DCL DD .WCBU0 CHAR(1) DEF(.WCBSWC) POS(1) ; .WCBDTA
00063          DCL DD .WCBU1 CHAR(1) DEF(.WCBSWC) POS(2) ; .WCBDTA
00064          DCL DD .WCBU2 CHAR(1) DEF(.WCBSWC) POS(3) ; .WCBDTA
00065          DCL DD .WCBU3 CHAR(1) DEF(.WCBSWC) POS(4) ; .WCBDTA
00066          DCL DD .WCBU4 CHAR(1) DEF(.WCBSWC) POS(5) ; .WCBDTA
00067          DCL DD .WCBU5 CHAR(1) DEF(.WCBSWC) POS(6) ; .WCBDTA
00068          DCL DD .WCBU6 CHAR(1) DEF(.WCBSWC) POS(7) ; .WCBDTA
00069          DCL DD .WCBU7 CHAR(1) DEF(.WCBSWC) POS(8) ; .WCBDTA
          ; .WCBDTA

00071          BRK *.PGT *                 /*PROCESS GLOBAL TABLE*/ ; .PGT
00072          DCL SPC .PGT BAS(.MGTPTGT)   ; .PGT
00073          DCL DD .PGLVL CHAR(16) DIR   ; .PGT

```

```

IBM S/38 COBOL S714C91      IRP LISTING FOR TESTPRT
SEQ  INST OFFSET  GENERATED CODE      *... 1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 ... 8 BREAK
00074          DCL SPCPTR .PGTMGT DIR      /*PTR TO FIRST MGT*/ ; .PGT
00075          DCL SPCPTR .PGTMGT DIR      /*PTR TO LAST MGT*/ ; .PGT
00076          DCL DD .PGTIND(32) CHAR(1) DIR /*INDICATORS*/ ; .PGT
00077          DCL DD .PGTINVC BIN(2) DIR    /*MAIN PGM INVOC #*/ ; .PGT
          ; .PGT
          /*1 - STOP STATEMENT EXECUTED*/ ; .PGT
          ; .PGT
00078          DCL DD * CHAR(10) DEF(.MGTNAME) INIT("TESTPRT ") ; .PGT
00079          DCL SPCPTR * DEF(.MGTFIB) INIT(.FOIFN) ; .PGT

00081          BRK *.INIT *                 ; .INIT
          /* INITIALIZATION CODE AND DECLARES*/ ; .INIT
          ; .INIT
00082          DCL DD .MPRCTMP BIN(4) BDRY(16) INIT(32) ; .INIT
00083          DCL DD * CHAR(12)            ; .INIT
00084          DCL SPCPTR .PASAPTR          /*-POINTER TO PASA*/ ; .INIT
00085          DCL SPCPTR .MPRCATR INIT(.MPRCTMP) ; .INIT
00086          DCL SPCPTR .PASACUR BAS(.PASAPTR) ; .INIT
00087          DCL SPCPTR .SEPTD           /*PTR TO SYS EP TBL */ ; .INIT
00088          DCL SYSPTR .MAINRRTN INIT("OCRMAIN",TYPE(PGM,1)) ; .INIT
00089          DCL INSPTR .DBUGRTN        /*RET PTR FROM DEBUG INCLD ; .INIT
          SE*/ ; .INIT
00090          DCL SPCPTR .MGTPTR INIT(.MGT) /*COBOL MODULE GLOBAL TABL ; .INIT
          E PTR*/ ; .INIT
00091          DCL DL .MGTST (.MGTPTGT) ; .INIT
00092          DCL CON *ON CHAR(1) INIT('1') /*INDICATOR ON VALUE*/ ; .INIT
00093          DCL CON *OFF CHAR(1) INIT('0') /*INDICATOR OFF VALUE*/ ; .INIT
00094          DCL CON *BATCH CHAR(1) INIT('B') /*BATCH JOB*/ ; .INIT
00095          DCL CON *INTER CHAR(1) INIT('I') /*INTERACTIVE JOB*/ ; .INIT
          ; .INIT
          /* FILE STATUS CODES */ ; .INIT
          ; .INIT
00098          DCL CON *FS00 CHAR(2) INIT('00') /*SUCCESSFUL COMPLETION*/ ; .INIT
00099          DCL CON *FS02 CHAR(2) INIT('02') /*DUPLICATE KEY-ALLOWED*/ ; .INIT
00100          DCL CON *FS10 CHAR(2) INIT('10') /*AT END*/ ; .INIT
00101          DCL CON *FS11 CHAR(2) INIT('11') /*NO DATA AVAILABLE*/ ; .INIT
00102          DCL CON *FS12 CHAR(2) INIT('12') /*NO MODIFIED SUBFILE REC*/ ; .INIT
          / ; .INIT
00104          DCL CON *FS21 CHAR(2) INIT('21') /*SEQUENCE ERROR*/ ; .INIT
00105          DCL CON *FS22 CHAR(2) INIT('22') /*DUP KEY-NOT ALLOWED*/ ; .INIT
00106          DCL CON *FS23 CHAR(2) INIT('23') /*NO RECORD FOUND*/ ; .INIT
00107          DCL CON *FS24 CHAR(2) INIT('24') /*BDRY VIOLTN-RELATIVE*/ ; .INIT
00108          DCL CON *FS30 CHAR(2) INIT('30') /*PERMANENT ERROR*/ ; .INIT
00109          DCL CON *FS34 CHAR(2) INIT('34') /*BOUNDARY VIOLATION*/ ; .INIT
00110          DCL CON *FS90 CHAR(2) INIT('90') /*UNSUCCESSFUL I/O*/ ; .INIT
00111          DCL CON *FS91 CHAR(2) INIT('91') /*INVALID ACCESS*/ ; .INIT
00112          DCL CON *FS92 CHAR(2) INIT('92') /*LOGIC ERROR*/ ; .INIT
00113          DCL CON *FS94 CHAR(2) INIT('94') /*NO CUR REC PTR - I/O*/ ; .INIT
00114          DCL CON *FS95 CHAR(2) INIT('95') /*INVLD/INCMPLT FIL INFO*/ ; .INIT
00115          DCL CON *FS96 CHAR(2) INIT('96') /*INPUT DATA TRUNCATED*/ ; .INIT
00116          DCL CON *FS91 CHAR(2) INIT('91') /*WRITE OP FAILED-TRANS*/ ; .INIT
00117          DCL CON *FS9K CHAR(2) INIT('9K') /*FRMT NAME INVLD/NOT FND*/ ; .INIT
00118          DCL CON *FS9H CHAR(2) INIT('9H') /*SUBFILE FULL*/ ; .INIT

```

Figure A-5 (Part 6 of 10). Examples of Compiler Debugging Information

```

IBM S/38 COBOL 5714CB1          IRP LISTING FOR TESTPRT
SEQ  INST OFFSET  GENERATED CODE          *... 1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 ... 8 BREAK
00163          ENTRY ,CALRTN INT                                ; ,CALEXCP
00164 000A 00003C 30B2 0012 0050  CPYBLA ,MGTTUFL,90N          ; ,CALEXCP
00165 000B 000042 03E1 00B4          RTNEXCP ,CALERP             ; ,CALEXCP
                                /*INTERNAL SIZE ERROR EXCEPTION HANDLER*/ ; ,CALEXCP
00166          BRK ,SIZEXCP*                                     ; ,CALEXCP
                                DCL EXCM ,SIZEXCM EXCID(H*0C03*,H*0C0A*,H*0C0B*) INT(.SIZERTN) IMD CV( ; ,SIZEXCP
                                X*00000000*) ; ,SIZEXCP
00169          DCL SPCPTR ,SIZERP INIT(.RTNPTR)                 ; ,SIZEXCP
00170          ENTRY ,SIZERTN INT                                ; ,SIZEXCP
00171 000C 000046 3143 400B 2001 2001 ADDN(5) ,MGTCNTR(1),1          /*-ADD 1 TO SIZE COUNT*/ ; ,SIZEXCP
00172 000D 00004E 10B2 400F 2001 0050 CPYBLA ,MGTIND(1),90N          /*-INDICATE ON SIZE ERROR*/ ; ,SIZEXCP
00173 000E 000056 10B2 400F 2002 0050 CPYBLA ,MGTIND(2),90N          ; ,SIZEXCP
00174 000F 00005E 03E1 00B7          RTNEXCP ,SIZERP             ; ,SIZEXCP
                                /*FUNCTION CHECK EXCEPTION MONITOR - CALLS QCREXHAN*/ ; ,SIZEXCP
00175          BRK ,FCEXCP*                                     ; ,FCEXCP
00176          DCL SYSPTR ,FCPTR INIT('QCREXHAN',TYPE(PGM))      ; ,FCEXCP
00177          DCL DD ,FCPARM CHAR(22) INIT                      ; ,FCEXCP
00178          DCL SPCPTR ,FCPARM INIT(.FCPARM)                 ; ,FCEXCP
00179          DCL DD ,FCLIST [,FCPARM,.,MGTPTR]                ; ,FCEXCP
00180          DCL EXCM ,FCEXCM EXCID(H*9999*) BP(.CPF9999) IMD CV(*CPF*) ; ,FCEXCP
00181          CALL ,FCEXCM EXCID(H*9999*) BP(.CPF9999)          /*CALL QCREXHAN WITH 2 PAR ; ,FCEXCP
00182 0010 000062 2283 00B9 00B0 0000 .CPF9999;CALLX ,FCLIST,* ; ,FCEXCP
                                MS*/ ; ,FCEXCP
00183 0011 00006A 1011 01F8          B ,RETURN                    /* GO DD STOP RUN FUNCTION ; ,FCEXCP
                                /* ; ,FCEXCP
                                /*STOP RUN EXCEPTION MONITOR*/ ; ,FCEXCP
00184          BRK ,STPEXCP*                                     ; ,STPEXCP
00185          DCL EXCM ,STPEXCM EXCID(H*9999*) BP(.RTX) IMD CV(*CBL*) ; ,STPEXCP
00186          BRK ,INIT002*                                     ; ,INIT002
00187          .INIT002CPYBWP ,RTNPTR,.,PASACUR                 ; ,INIT002
00188 0012 00006E 2132 007E 00AA          .INIT002CPYBWP ,RTNPTR,.,PASACUR ; ,INIT002
00189 0013 000074 1042 0079 2001          CPYV ,PERFCTR,1           /*INIT PERFORM NEST CTR*/ ; ,INIT002
00190          BRK ,DICT*                                       ; ,DICT
                                LARES */ ; ,DICT
                                /*FIB DSECT (FILE INFORMATION BLOCK)*/ ; ,DICT
00191          DCL INSPTR ,USERTN                                ; ,DICT
                                /* USE PROCEDURE RTNPOINTE ; ,DICT
                                R */ ; ,DICT
00192          DCL INSPTR ,FIBUSAV                               ; ,DICT
                                /* USE SAVE POINTER */ ; ,DICT
00193          DCL INSPTR ,PRODM BAS(.FIBPTR)                   ; ,DICT
                                /*PROCEDURE RTNPOINTE */ ; ,DICT
00194          DCL SPCPTR ,FIBPTR                                ; ,DICT
                                /*.FIB BASE*/ ; ,DICT
00195          DCL SPC ,FIB BAS(.FIBPTR)                        ; ,DICT
00196          DCL DD ,FIBFN CHAR(30) DIR                       ; ,DICT
                                /*FD NAME*/ ; ,DICT
00197          DCL DD ,FIBALT CHAR(1) DIR                       ; ,DICT
                                /*ALTERNATE ID FLAGS*/ ; ,DICT
00198          /* BIT 1 CLEAR FILE */ ; ,DICT
00199          /* BIT 2 INITIALIZE FILE TO DELETED RECORDS */ ; ,DICT
00200          /* BIT 3 RESET CURRENT RECORD POINTER IN EFFECT */ ; ,DICT
00201          /* SET ON BY DELETE AND REWRITE. SET OFF BY OPEN,READ,START.*/ ; ,DICT

```

```

IBM S/38 COBOL 5714CB1          IRP LISTING FOR TESTPRT
SEQ  INST OFFSET  GENERATED CODE          *... 1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 ... 8 BREAK
                                /*FILE CONTROL BLOCK*/ ; ,DICT
00489          BRK ,F02*                                       ; ,DICT
00490          DCL DD ,F02FN CHAR(30) JDRY(16) INIT('FILE-2    ') ; ,F02
00491          /* FD NAME */ ; ,F02
00492          DCL DD ,F02ALTS CHAR(1)                           ; ,F02
                                /*NBR ALT KEYS*/ ; ,F02
00493          DCL DD ,F02FLGS CHAR(1) INIT('X00')              ; ,F02
                                /*BITS - OPEN,LOCKED,EOF,M ; ,F02
                                /*OPTIONAL*/ ; ,F02
00494          DCL DD ,F02OLD CHAR(6) INIT('X00000000F0F0') /*LAST OP AND STATUS*/ ; ,F02
00495          DCL DD ,F02CUR CHAR(6) INIT('X00000000F0F0') ; ,F02
00496          DCL DD ,F02COP CHAR(4) DEF(.F02CUR) PDS(1)       ; ,F02
                                /*CUR OP AND STATUS*/ ; ,F02
00497          DCL DD ,F02USE# BIN(2) INIT                       ; ,F02
                                /*USE PARA*/ ; ,F02
00498          DCL DD ,F02P# CHAR(10) INIT                      ; ,F02
                                /*PRT IN EFFECT*/ ; ,F02
00499          DCL DD ,F02SPC CHAR(14) INIT('X00000000000000000000000000000000') ; ,F02
                                /*SPECIAL DATA*/ ; ,F02
00500          DCL DD * CHAR(10)                                  ; ,F02
                                /*FILLER*/ ; ,F02
00501          DCL SPCPTR ,F02UFCB INIT(.U02UFCB)              ; ,F02
                                /*PTR TO UFCB*/ ; ,F02
00502          DCL INSPTR ,F02UBD0                              ; ,F02
                                /*USE BRANCH TO POINTER */ ; ,F02
00503          DCL SPCPTR ,F02UJYN                               ; ,F02
                                /*USE RETURN POINTER */ ; ,F02
00504          DCL SPCPTR ,F02CHAN                               ; ,F02
                                /*NULL PTR TO FIB*/ ; ,F02
                                /*UFCB FOR THE FILE */ ; ,F02
00505          DCL SPCPTR ,U02UFCB                               ; ,F02
                                /*PTR TO DDPCR*/ ; ,F02
00506          DCL SPCPTR ,U02IBF#                               ; ,F02
                                /*PTR TO INPUT BUF*/ ; ,F02
00507          DCL SPCPTR ,U02OBF#                               ; ,F02
                                /*PTR TO OUTPUT BUF*/ ; ,F02
00508          DCL SPCPTR *                                       ; ,F02
                                /*OPEN FEEDBACK*/ ; ,F02
00509          DCL SPCPTR *                                       ; ,F02
                                /*I-O FEEDBACK*/ ; ,F02
00510          DCL SPCPTR *                                       ; ,F02
                                /*PTR TO NEXT UFCB TO OPEN ; ,F02
                                EDS*/ ; ,F02
00511          DCL DD * CHAR(32)                                 ; ,F02
00512          DCL DD * CHAR(10) INIT('MASTER    ')             ; ,F02
                                /*FILE NAME*/ ; ,F02
00513          DCL DD * BIN(2) INIT(~75)                         ; ,F02
                                /*LIBRARY*/ ; ,F02
00514          DCL DD * CHAR(10) INIT('MLIBL')                   ; ,F02
00515          DCL DD * BIN(7) INIT(~71)                         ; ,F02
                                /*MEMBER*/ ; ,F02
00516          DCL DD * CHAR(10) INIT                    ; ,F02
00517          DCL DD * CHAR(10) INIT                    ; ,F02
                                /* NAME OF DEVICE */ ; ,F02
00518          DCL DD * BIN(2) INIT                    ; ,F02
                                /*INDEX OF DEVICE IN DD*/ ; ,F02
00519          DCL DD ,U02FLGS CHAR(2) INIT('X0000')           ; ,F02
00520          DCL DD * CHAR(4) INIT('0100')                    ; ,F02
                                /*REL AND VER*/ ; ,F02
00521          DCL DD * CHAR(28)                               ; ,F02
                                /*RESERVED*/ ; ,F02
00522          /*END OF COMMON PARM OF UFCB*/ ; ,F02
00523          /* OPTION - PRIMARY RECORD LENGTH */ ; ,F02
00524          DCL DD * BIN(2) INIT(01)                           ; ,F02
00525          DCL DD * BIN(2) INIT(00020)                       ; ,F02
                                /* LENGTH OF RECORD */ ; ,F02
00526          /* OPTION - INHIBIT PAGE OVERFLOW*/ ; ,F02
00527          DCL DD * BIN(2) INIT(20)                           ; ,F02
00528          DCL DD * CHAR(1) INIT('X*80')                     ; ,F02
                                /*INHIBIT = YES*/ ; ,F02
00529          /* OPTION - DEVICE DEPENDENT OR CONTROL LIST USED*/ ; ,F02
00530          ; ,F02

```

Figure A-5 (Part 7 of 10). Examples of Compiler Debugging Information

```

IBM S/38 COBOL 5714CB1          IRP LISTING FOR TESTPRT
SEQ  INST OFFSET  GENERATED CODE          *... 1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 ... 8 BREAK
00531          DCL DD * BIN(2) INIT(03)          ; :F02
00532          DCL DD * CHAR(1) INIT('X*80")          /*DEVICE CONTROLS USED*/ ; :F02
00533          DCL DD * BIN(2) INIT(58)          ; :F02
00534          DCL DD * CHAR(1) INIT('X*80")          ; :F02
00535          DCL DD * BIN(2) INIT(10)          /*INDICATE BLOCKING DESIRE ; :F02
          ; :F02
          ; :F02
          /* OPTION - LEVEL CHECK NO */ ; :F02
          ; :F02
00536          DCL DD * BIN(2) INIT(06)          ; :F02
00537          DCL DD * UO2LVCK CHAR(1)          /*LEVEL CHECK NO */ ; :F02
00538          DCL DD * UO2LVAL CHAR(1) DEF(,UO2LVCK) INIT('X*00") ; :F02
00539          DCL DD * CHAR(2) INIT('X*7FFF") ; :F02
00540          /*-- END OF UF08 --*/ ; :F02
          ; :F02
          ; :DICT
00541          BRK *DICT* ; :DICT
00542          DCL DD *F02BUFR CHAR(00020) BAS(,UO2BUFR) LVL('FD') HLL('FILE-2') ; :DICT
00543          DCL DD *D0054E2 CHAR(00020) DEF(,F02BUFR) POS(1) LVL('01') HLL( ; :DICT
          *RECORD2*) PARENT('FILE-2','FD') ; :DICT
00544          DCL DD *D005536 CHAR(00020) DEF(,F02BUFR) POS(1) LVL('02') HLL( ; :DICT
          *FIELD*) PARENT('RECORD2','01') ; :DICT
00545          DCL DD *D005580 CHAR(00056) BDRY(2) LVL('01') HLL('FILLER') ; :DICT
00546          DCL DD *D005502 PKD(02,00) DEF(,D005580) POS(1) LVL('05') HLL('KOUNT') ; :DICT
00547          PARENT('FILLER','01') ; :DICT
00548          DCL DD *D005574 CHAR(00026) DEF(,D005580) POS(3) LVL('05') HLL( ; :DICT
          *ALPHABET) PARENT('FILLER','01') INIT('ABCDEFGHIJKLMNOPQRSTUVWXYZ') ; :DICT
00549          DCL DD *A005680(26) CHAR(00001) DEF(,D005574) LVL('05') HLL('ALPHA') ; :DICT
          PARENT('FILLER','01') ; :DICT
00550          DCL DD *D005680 CHAR(00001) DEF(,BSTRING) POS(1) ; :DICT
00551          DCL DD *D005710 PKD(02,00) DEF(,D005580) POS(29) LVL('05') HLL('NUM') ; :DICT
          ) PARENT('FILLER','01') ; :DICT
00552          DCL DD *D005772 CHAR(00026) DEF(,D005580) POS(31) LVL('05') HLL( ; :DICT
          *NO-OF-DEPENDENTS*) PARENT('FILLER','01') INIT( ; :DICT
          *01000012000123001234002340") ; :DICT
00553          DCL DD *A005776(26) CHAR(00001) DEF(,D005772) LVL('05') HLL('DEPEND') ; :DICT
          PARENT('FILLER','01') ; :DICT
00554          UCL DD *D0057F6 CHAR(00001) DEF(,BSTRING) POS(1) ; :DICT
00555          DCL DD *D005856 CHAR(00019) BDRY(2) LVL('01') HLL('WORKRECORD') ; :DICT
00556          DCL DD *D0058AC CHAR(00001) DEF(,D005856) POS(1) LVL('05') HLL( ; :DICT
          *NAMES-FIELD*) PARENT('WORKRECORD','01') ; :DICT
00557          DCL DD *D00590A CHAR(00001) DEF(,D005856) POS(2) LVL('05') HLL( ; :DICT
          *FILLER*) PARENT('WORKRECORD','01') INIT ; :DICT
00558          DCL DD *D00595E ZND(03,00) DEF(,D005856) POS(3) LVL('05') HLL( ; :DICT
          *RECORDNO*) PARENT('WORKRECORD','01') ; :DICT
00559          DCL DD *D0059C2 CHAR(00001) DEF(,D005856) POS(6) LVL('05') HLL( ; :DICT
          *FILLER*) PARENT('WORKRECORD','01') INIT ; :DICT
00560          DCL DD *D005A1C CHAR(00003) DEF(,D005856) POS(7) LVL('05') HLL( ; :DICT
          *LOCATION*) PARENT('WORKRECORD','01') INIT('NYC') ; :DICT
00561          DCL DD *D005A82 CHAR(00001) DEF(,D005856) POS(10) LVL('05') HLL( ; :DICT
          *FILLER*) PARENT('WORKRECORD','01') INIT ; :DICT
00562          DCL DD *D005A9C CHAR(00002) DEF(,D005856) POS(11) LVL('05') HLL( ; :DICT
          *DEPENDENTS*) PARENT('WORKRECORD','01') ; :DICT
00563          DCL DD *D005B32 CHAR(00007) DEF(,D005856) POS(13) LVL('05') HLL( ; :DICT
          *FILLER*) PARENT('WORKRECORD','01') INIT ; :DICT
00564          DCL DD *D005B8C PKD(02,00) BDRY(2) LVL('01') HLL('NODEPEND') ; :DICT
00565          DCL DD *D005BEE ZND(02,00) BDRY(2) LVL('01') HLL('ZONE-DATA') ; :DICT

```

```

IBM S/38 COBOL 5714CB1          IRP LISTING FOR TESTPRT
SEQ  INST OFFSET  GENERATED CODE          *... 1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 ... 8 BREAK
00566          DCL DD *V005BEE CHAR(00001) DEF(,D005BEE) POS(2) ; :DICT
00567          DCL DD *D005CAA CHAR(00010) LVL('01') HLL('DB-FORMAT-NAME') ; :DICT
00568          DCL DD *D00707C CHAR(00029) INIT('EMPLOYEES WITH NO DEPENDENTS ') ; :DICT
00569          DCL DD *PT BDRY(16) CHAR(0006A)          /* PARAGRAPH INITIALIZATIO ; :DICT
          N TABLE */ ; :DICT
00570          DCL DD *PTHDR CHAR(16) DEF(,PT) POS(1) ; :DICT
          /* PARAGRAPH TABLE HEADER ; :DICT
          */ ; :DICT
00571          DCL DD * CHAR(7) DEF(,PTHDR) POS(1) INIT('PT 01.01') ; :DICT
          /* NAME AND LEVEL */ ; :DICT
00572          DCL DD * BIN(2) DEF(,PTHDR) POS(8) INIT(16) /* SIZE OF HEADER */ ; :DICT
00573          DCL DD *PTNUM BIN(2) DEF(,PTHDR) POS(10) INIT(0001) ; :DICT
          /* NO. OF ENTRIES */ ; :DICT
00574          DCL DD *PTSEG CHAR(11) DEF(,PTHDR) POS(12) INIT('01') ; :DICT
00575          DCL DD *PTABLE CHAR(00048) DEF(,PT) POS(17) /* TABLE OF PROC PTRS */ ; :DICT
00576          DCL INSPTR *PBP000A DEF(,PTABLE) POS(00001) ; :DICT
00577          DCL INSPTR * DEF(,PTABLE) POS(00017) INIT(,PAR0005) ; :DICT
00578          DCL DD * BIN(2) DEF(,PTABLE) POS(00033) INIT(00004) ; :DICT
00579          DCL SPGPTD *TADR001          /* ADDRESS TEMPORARY AND ; :DICT
          /* BUFFER FOR DISPLAY AND ; :DICT
          ACCEPT */ ; :DICT
00580          DCL DD *TC00001 CHAR(00056) ; :DICT
00581          DCL DD *TC00002 CHAR(00056)          /* CHAR TEMPS FOR IF */ ; :DICT
00582          DCL DD *SVOBFR CHAR(00056) ; :DICT
00583          DCL DD *PRBUFR CHAR(00056) BAS(,UCBOBR) ; :DICT
00584          DCL DD *CSEPSGN CHAR(2) INIT('+-')          /* SIGNS FOR IF */ ; :DICT
00585          DCL DD *CIMBSGN CHAR(60) INIT( ; :DICT
          X'F0F1F2F3F4F5F6F7F8F90001D2D3D4D5D6D7D8D9A0A1A2A3A4A5A6A7A8A9B0B1B2B3 ; :DICT
          B4B5B6B7B8B9C0C1C2C3C4C5C6C7C8C9E0E1E2E3E4E5E6E7E8E9') ; :DICT
00586          DCL DD *CALPHAB CHAR(27) INIT(' ETADINSHRDLUCMFPYVWGKJXZ') ; :DICT
          /* HEX NUMERICS FOR IF */ ; :DICT
          /* NUMERICS FOR IF */ ; :DICT
          /* ALPHABETICS FOR IF */ ; :DICT
          /* PAD CHAR FOR IF */ ; :DICT
00589          DCL DD *CPADCHR CHAR(1) INIT(' ') ; :DICT
00590          B *START ; :DICT
00591          ; :DICT
00592          ; :DICT
00593          ; :DICT
00594          BRK *51" ; :DICT
00595          SETSP *FIBPTR,,FOIPN          /*OPEN */ ; 51
00596          CPYBLA *FIBALT,X*80"          /*POINT TO CFCB*/ ; 51
00597          CPYNV *FIBUSE,,0          /*CLEAR FILE*/ ; 51
00598          CPYBLA *U0IFLGS,X*0010"          /*NO USE SPECIFIED*/ ; 51
00599          CALLI *OPEN *,,IORTN          /*SET OPEN OUTPUT*/ ; 51
00600          CPYBLA *FIBKFM,*          /*CALL WITH RETURN POINT*/ ; 51
00601          CPYNV *FIBVERB,,0          /*BLANK FORMATTAMES*/ ; 51
00602          BRK *52"          /*SAVE VERB TYPE*/ ; 51
00603          CPYNV *D0055D2,0          /*MOVE */ ; 52
00604          CPYNV *D005710,0          /*COPY SOURCE TO RECV */ ; 52
00605          CPYNV *D00588C,0          /*COPY SOURCE TO RECV */ ; 52
00606          ; :DICT
00607          BRK *53"          /*ADD */ ; 53
00608          ADDN(5) *D0055D2,1          /*ADD */ ; 54
00609          BRK *54"          /*ADD */ ; 54
00610          ADDN(5) *D005710,1          /*ADD */ ; 54
00611          BRK *55"          /*MOVE */ ; 55
00612          CPYNV *BINSUB,,D0055D2          /*MOVE */ ; 55
          /*COPY SOURCE TO RECV */ ; 55
          SETSP *TADR001,,A005680(,BINSUB ) ; 55

```

Figure A-5 (Part 8 of 10). Examples of Compiler Debugging Information

SEQ	INST	OFFSET	GENERATED CODE	IRP LISTING FOR TESTPRT	BREAK
00613	0023	0000D6	10B3 0184 81AE 01C8 2040	CPYBLAP .D0059AC.,TADR001->.D0056B0. " "	55
00614				BRK #56 " /%MOVE %/	56
00615	0024	0000E0	1042 0078 01AB	CPYVW .BINSUB..D0055D2	56
00616	0025	0000E6	0082 01C8 41B1 0078	SETSP .TADR001..A0057F6(.BINSUB)	56
00617	0026	0000EE	10B3 018A 81B2 01C8 2040	CPYBLAP .D005ADC.,TADR001->.D0057F6. " "	56
00618				BRK #57 " /%MOVE %/	57
00619	0027	0000F8	1042 01B6 01AF	CPYVW .D00595E.,D005710	57
00620				.PAR0004: /%COPY SOURCE TO RECVR %/	57
00621				BRK #58 " /%WRITE %/	58
00622	0028	0000FE	30B3 0179 01B3 2040	CPYBLAP .D00539A.,D005856. " "	58
00623	0029	000106	0082 0023 014C	SETSP .FIBPTR.,F01FN	58
00624	002A	00010C	1022 011C 01D0	CPYBLA .FIB0P.X*0000005"	58
00625	002B	000112	0082 0126 011A	SETSP .IDCPTR.,NULLCL	58
00626	002C	000118	0293 020F 0000 0022	CALL I .PUTALL.,.IORTN	58
00627	002D	000120	3CC2 C000 0090 005C 01DC	CMPLAIB) .FIBCF5,%F524/NEQ1L000001	58
00628	002E	00012A	10B2 009D 005E	CPYBLA .FIBCF5,%F534	58
00629				.L000001: /%BOUNDARY VIOLATION%/	58
00630	002F	000130	3042 00A2 2007	CPYVW .FIBVERB.,07	58
00631	0030	000136	0083 0117 0000 0119	ADDSPP .DMPCDFP.,UCBIDF.,DMPCDFD	58
00632	0031	00013E	10B3 01BF 0143 2040	CPYBLAP .D005CAA.,DMPRFMT. " "	58
00633	0032	000146	1011 01C8	II .PBP0004	58
00634				.PAR0005: /%COPY FORMAT NAME%/	58
00635				BRK #59 " /%PERFORM %/	59
00636				.L000001: /%BOUNDARY VIOLATION%/	59
00637	0033	00014A	3C46 C000 01AB 201A 01DF	CMPNV(B) .D0055D2,26/NEQ1L000004	59
00638	0034	00015A	1011 01E3	B .L000002	59
00639				.L000004: /%BOUNDARY VIOLATION%/	59
00640				DCL INSPTR .IP00001	59
00641	0035	00015B	2132 01E0 01C8	CPYBWP .IP00001.,PBP0004	59
00642	0036	00015E	1022 01C8 01E1	SETIP .PBP0004.,L000004	59
00643	0037	00016A	1011 01D9	B .PAR0003	59
00644	0038	000168	2132 01C8 01E0	.L000004:CPYBWP .PBP0004.,IP00001	59
00645				.L000003: /%RESTORE OLD BR PTR%/	59
00646	0039	00016E	3011 01DE	B .L000001	59
00647				.L000002: /%BOUNDARY VIOLATION%/	59
00648				.PAR0006: /%BOUNDARY VIOLATION%/	59
00649				BRK #60 " /%CLOSE %/	60
00650	003A	000172	2082 0023 014C	SETSP .FIBPTR.,F01FN	60
00651	003B	000178	0132 00D7 00A4	CPYBWP .UFCBPR.,FIBUF5C	60
00652	003C	00017E	1197 00EA 2080	DR (5) .UCBFLG1.X*80"	60
00653	003D	000184	0293 0209 0000 0022	CALL I .CLOSE.,.IORTN	60
00654	003E	00018C	3042 00A2 2001	CPYVW .FIBVERB.,01	60
00655	003F	000192	10B2 01BF 2040	CPYBREP .D005CAA. " "	60
00656				BRK #61 " /%BLANK FORMATNAME%/	61
00657	0040	000198	0082 0023 0178	SETSP .FIBPTR.,F02FN	61
00658	0041	00019E	1042 009F 2000	CPYVW .FIBUSE.,0	61
00659	0042	0001A8	1082 0197 01E5	CPYBLA .U02FLGS.X*0020"	61
00660	0043	0001AA	0293 01FE 0000 0022	CALL I .OPEN.,.IORTN	61
00661	0044	0001B2	30B2 00BE 2040	CPYBREP .FIBCFMT. " "	61
00662	0045	0001B8	1042 00A2 2003	CPYVW .FIBVERB.,03	61
00663				.PAR0007: /%SAVE VERB TYPE%/	61
00664				BRK #62 " /%READ %/	62

SEQ	INST	OFFSET	GENERATED CODE	IRP LISTING FOR TESTPRT	BREAK
00714	0063	000280	1042 01F4 01BC	CPYVW .T000003.,D005B8C	69
00715	0064	000286	1042 01F1 2002	CPYVW .DISPPDS,00002	69
00716	0065	00028C	1143 006E 2002	ADDN(S) .ADLN,00002	69
00717	0066	000292	0083 01F0 01F0 01F1	ADDSPP .DISPTR.,DISPTR.,DISPPDS	69
00718	0067	00029A	0082 006A 01CC	SETSP .ADBUF.,ADBUFLV	69
00719	0068	0002A0	10B2 006B 20C4	CPYBLA .ADFUNC.,0D	69
00720	0069	0002A6	1042 006C 2009	CPYVW .ADTYP,00009	69
00721	006A	0002AC	1042 001D 2001	CPYVW .MGTLVL,1	69
00722	006B	0002B2	0082 002A 006A	SETSP .MGTPARM.,ADBUF	69
00723	006C	0002B8	0283 01F2 004F 0000	CALL X .DCRDISP.,MGTSTL,*	69
00724				BRK #70 " /%ADD %/	70
00725				DCL DD .Y010200 ZND(02,00)DEFI.TMPN01)	70
00726	006D	0002C0	1042 01F5 01BD	CPYVW .Y010200.,D005BEE	70
00727	006E	0002C6	1143 01F5 2001	ADDN(S) .Y010200,1	70
00728	006F	0002CC	1052 01BD 01F5	EXTRAG .D005BEE.,Y010200	70
00729				BRK #71 " /%STOP %/	71
00730	0070	0002D2	1042 0019 2001	CPYVW .MGTFUNC,1	71
00731	0071	0002D8	0283 004C 004F 0000	CALL X .MAINRTN.,MGTSTL,*	71
00732				.L000018: /%STOP RUN FUNCTION%/	71
00733				.L000019: /%CODE FOR CBL9999 EXCEPTI	71
00734	0072	0002E0	3042 0019 2001	.RETURN: CPYVW .MGTFUNC,1	71
00735	0073	0002E6	0283 004C 004F 0000	CALL X .MAINRTN.,MGTSTL,*	71
00736				DCL DD .RCLM CHAR(1) INITI(C*)	71
00737				DCL SPCPTR .RCLM INITI(.RCLM)	71
00738				DCL OL .RCLMQL(.RCLM)	71
00739	0074	0002EE	2225 0000	.RTX: DEACTPG *	71
00740	0075	0002F2	0283 402D 2056 01FB 0000	CALL X .WCBSEPT(86) .RCLMQL,*	71
00741	0076	0002FC	02A1 0000	RTX *	71
00742				/S----- COMMON INCLUDES ----- S/	71
00743				DCL DD .UFLGSAV CHAR(2)	71
00744				BRK #.OPEN " /%SAVE OLD UFCB FLAG%/	71
00745				ENTRY .OPEN INT /%ENTRY POINT FOR OPEN%/	71
00746	0077	000300	30B2 0098 0078	CPYBLA .FIBOLD.,FIBCUR	71
00747	0078	000306	10B2 009D 0054	CPYBLA .FIBCF5,%F500	71
00748	0079	00030C	1C2A 4000 0097 20B0 01FF	TSTBUM(B) .FIBFLGS.X*80"/ZER(.OPEN001)	71
00749	007A	000316	10B2 009D 0061	CPYBLA .FIBCF5,%F592	71
00750	007B	00031C	1011 0022	R .IORTN	71
00751				.OPEN001: /%SET ERROR STATUS%/	71
00752	007C	000320	3C2A 4000 0097 2040 0500	TSTBUM(R) .FIBFLGS.X*40"/ZER(.OPEN002)	71
00753	007D	00032A	10B2 009D 0061	CPYBLA .FIBCF5,%F592	71
00754	007E	000330	1011 0022	R .IORTN	71
00755				.OPEN002: /%RETURN%/	71
00756	007F	00033A	3193 0097 205F	AND(S) .FIBFLGS.X*5F"	71
00757	0080	00033A	0132 00D7 00A4	CPYBWP .UFCBPR.,FIBUF5C	71

Figure A-5 (Part 9 of 10). Examples of Compiler Debugging Information

OFFSET	MI	TEMPLATE	DISPLAY	IRP	LISTING	FOR	TESTPRT
00000000	000045F	00000000	0201E3C5	E2E3D7D9	E3404040	40404040	40404040
00000020	40404040	40404040	C0000000	00000000	00000000	00010000	00000000
00000040	00000000	00000000	000A0C5C	57000400	00000000	00000000	00000000
00000060	000000FC	00000000	00000000	00DF022A	00000100	000006CA	00000F70
00000080	0000G21B	00004244	00000000	00002642	00001C00	00000000	00000000
000000A0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000C0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000E0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000100	00000SC4	23330049	0000201D	01320006	00A1042	00192002	0283004C
00000120	1082006D	00181093	0072000C	20401011	00903082	00130050	03E10081
00000140	005003E1	00843143	40082001	200110B2	400F2001	00501082	400F2002
00000160	00872283	0089008C	00001011	01F82132	007E004A	10420079	20011011
00000180	0223014C	10820076	20801042	009F2000	10820168	01085293	01FC0000
000001A0	008E2040	104200A2	20031042	01A82000	10A201AF	200010A2	01822000
000001C0	20011143	01AF2001	10420078	01AB0082	01CB41AD	007810H3	01R481AE
000001E0	10420078	01AB0082	01CB41B1	007810B3	01R481B2	01CB2040	104201B6
00000200	017901B3	20400082	0023014C	1082011C	01080082	0126011A	0293020F
00000220	3C2C2000	0090005C	01DC1082	009D005E	304200A2	200700B3	011700DD
00000240	01BF0143	20401011	01C83C46	C0001A85	201A010F	101101E3	213201E0
00000260	01C801E1	101101D9	213201C8	01E03011	01DE2082	0023014C	013200D7
00000280	00E42080	02930209	00000022	304200A2	2001108E	01BF2040	00820023
000002A0	009F2000	10820197	01E50293	01FE0000	0022308E	008E2040	104200A2
000002C0	00230178	1082011C	01E70082	0126011A	0293020D	00000022	304200A2
000002E0	011700DD	01191083	018F9143	20401C22	C000009E	20F101E8	101101E8
00000300	308301B3	01E92040	3CC3C000	018A20F0	01D051EE	108301B8	20E92040
00000320	018C1143	01ED2001	1052018C	01ED3011	01E62082	00230178	013200D7
00000340	00E42080	02930209	00000022	304200A2	2001108E	01BF2040	008201F0
00000360	008E2000	108301F3	01C02040	104201F1	201D1143	006E2010	008301F0
00000380	104201F4	018C1042	01F12002	1143005E	200200B3	01F001F0	01F10082
000003A0	10820068	20C41042	006C2009	1042001D	20010082	002A006A	028301F2
000003C0	104201F5	018D1143	01F52001	1052018D	01F51042	00192001	0283004C
000003E0	30420019	20010283	004C004F	00002225	00000283	40202056	01F80000
00000400	30820098	00981082	00900054	1C2A4000	00972080	01F10082	00900061
00000420	3C2A4000	00972040	02001082	009D0051	10110022	31930097	205F0132
00000440	1C2A4000	00962080	02031082	01FD00E9	02830202	02010000	1CC2A000
00000460	00221082	00E901FD	2132000E	00000283	402D200C	01280000	1C2A4000
00000480	02041CC2	40000105	20C40206	10110205	3CC2A000	010520E4	02061CC2
000004A0	20000206	3082009D	00633C2A	200000E8	203C0207	1C2A2000	00E82018
000004C0	200000E8	20300207	1C2A2000	00E82010	02070132	0008000A	10110208
000004E0	00972010	02080132	00DA00D8	31970097	20801011	00223082	00980098
00000500	020A1082	009D0054	1C2A2000	00972080	020C1082	00906208	10110022
00000520	00A40132	00DE0000	0283A02D	20080128	00001197	00972020	11930097
00000540	00222132	00D700A4	01320114	00D91082	00980098	1082009D	00541C2A
00000560	2008020E	1C2A2000	0110200E	020E1082	01E2003	3082009C	011C0083
00000580	01480132	012A402D	01310082	0125009C	02830124	01270000	013200D8
000005A0	00222132	00D700A4	01320114	00D91082	00980098	1082009C	011C1082
000005C0	00830129	01140148	01320124	402D0135	00820125	009C0283	01240127
000005E0	00223C2A	20000097	20010217	11970097	20201082	009D0056	10110022
00000600	00981011	00223C2A	20000097	2008021A	1082009D	005F109E	02102040
00000620	02140000	1C464000	009F2000	01FC1011	00223197	00972020	10110022
00000640	005A1011	00223082	009D005E	10110022	3082009D	00661011	00223082
00000660	308E0210	20400283	02150214	00001CC2	40000211	20C40226	1CC2C000
00000680	02261082	009D005C	3C464000	009F2000	01FC1011	00222132	02280023
000006A0	809300A6	10228093	00A6022A	101100A5	21320023	02280132	809300A6
000006C0	00910260	000008AC	20011001	08A40004	10010000	10010000	10010000
000006E0	10910000	10010000	10020000	10040000	00800017	0004000A	00080002

Figure A-5 (Part 10 of 10). Examples of Compiler Debugging Information

Appendix B. Summary of IBM Extensions

This appendix contains a brief summary of the extensions to System/38 COBOL with references to discussions of the extensions elsewhere in this manual.

Character-String Considerations

The maximum length of a non-numeric literal is 160 characters. See “Character-Strings, Literals” in Chapter 2.

Identification Division

The system uses the first 10 characters of the program-name specified in the PROGRAM-ID paragraph. See “PROGRAM-ID Paragraph” in Chapter 3.

Environment Division

The DEV parameter of an Override command can change the device type that the file will use. See “FILE-CONTROL Paragraph, Assign Clause” in Chapter 3.

FORMATFILE must be specified in the ASSIGN clause to use an externally described printer file. See “FILE-CONTROL Paragraph, Assign Clause” in Chapter 3.

COBOL programs can process data base files. The ORGANIZATION of the file indicates the current program usage. See “File Processing Summary” and “FILE-CONTROL Paragraph, ORGANIZATION Clause” in Chapter 3.

If the DDS keyword DESCEND is used when the field is specified as a key, the sequence can be a descending key sequence. The OVRDBF command can set the current record pointer when the file is opened. See “FILE-CONTROL Paragraph, ACCESS MODE Clause” in Chapter 3.

The RECORD KEY data item, data-name-2, can be numeric. EXTERNALLY-DESCRIBED-KEY can be specified in the RECORD KEY clause. See “FILE-CONTROL Paragraph, RECORD KEY Clause (Indexed File)” in Chapter 3.

The DUPLICATES phrase can be specified for the RECORD KEY clause. See “FILE-CONTROL Paragraph, RECORD KEY Clause (Indexed File)” in Chapter 3. Additional information is given in discussions of the READ, REWRITE, DELETE, and WRITE statements in Chapter 5.

The keywords specified for the data item in DDS can modify record sequence. See “FILE-CONTROL Paragraph, RECORD KEY Clause (Indexed File)” in Chapter 3.

The COMMITMENT CONTROL clause can be specified to enable the synchronizing or cancelling of data base changes, and to provide additional record locking for records being changed. See “I-O-CONTROL Paragraph, COMMITMENT CONTROL Clause” in Chapter 3 and and “Commitment Control Considerations” in Chapter 9.

Data Division

Elementary items or group items immediately subordinate to one group item can have unequal level-numbers. See “Data Description Concepts, Level-Numbers” in Chapter 4.

The OVRTAPF command can change the LABEL RECORDS clause at execution time. See “File Description Entry, LABEL RECORDS Clause” in Chapter 4.

If the CODE-SET clause is omitted, the CODE parameter of the CRTDKTF or the CRTTAPF command is used. The OVRDKTF or the OVRTAPF command can change the CODE-SET clause at execution time. See “File Description Entry, CODE-SET Clause” in Chapter 4.

For the USING phrase of the CALL statement, data-names can have level-numbers that are not 01 or 77, and the data-names can be indexed, subscripted, or qualified. See “CALL Statement, USING Phrase” in Chapter 6.

A FILLER item can be used as a group item definition. See “Data-Name or FILLER Clause” in Chapter 4.

COMPUTATIONAL-3 (packed decimal) and COMPUTATIONAL-4 (binary) can be specified for the USAGE clause of numeric items. See “USAGE Clause” in Chapter 4.

The key specified for an OCCURS clause can have USAGE of COMPUTATIONAL-3 or COMPUTATIONAL-4. See “OCCURS Clause” in Chapter 6.

The JUSTIFIED clause can be specified for alphanumeric edited items. See “JUSTIFIED Clause” in Chapter 4.

Procedure Division

The mnemonic-names OPEN-FEEDBACK and I-O-FEEDBACK are used for file information and are accessed through an ACCEPT statement format. See “ACCEPT Statement” in Chapter 5.

THEN is used as a separator on the IF statement. See “IF Statement” in Chapter 5.

The AT END phrase can be omitted for the READ statement. See “Common Input/Output Phrases, INVALID KEY Condition” in Chapter 5.

The FORMAT phrase is valid for DELETE, READ, REWRITE, START, and WRITE statements. See “DELETE Statement”, “READ Statement”, “REWRITE Statement”, “START Statement”, and “WRITE Statement” in Chapter 5.

The special register DB-FORMAT-NAME contains information about execution of file input/output statements. See “Common Input/Output Phrases, DB-FORMAT-NAME Special Register” in Chapter 5.

The identifier in an ACCEPT statement can have USAGE of COMPUTATIONAL-3 or COMPUTATIONAL-4. See “ACCEPT Statement” in Chapter 5.

The ACCEPT statement can be used to transfer data from a job’s local data area to a specified data item. See “ACCEPT Statement” in Chapter 5.

The system always rewinds and unloads the tape when REEL/UNIT is specified in the CLOSE statement. See “CLOSE Statement” in Chapter 5.

The INHWRT parameter of the OVRDBF command can inhibit the DELETE, READ, and WRITE statements. See “DELETE Statement”, “READ Statement”, and “WRITE Statement” in Chapter 5.

For a file with duplicate primary keys allowed, a READ statement must immediately precede a DELETE or REWRITE statement to ensure proper deletion. See “DELETE Statement” and “REWRITE Statement” in Chapter 5.

For the DISPLAY statement, COMPUTATIONAL-4 items are converted to zoned decimal items and signed noninteger numeric literals are allowed. See “DISPLAY Statement” in Chapter 5.

The DISPLAY statement can be used to transfer data to a job’s local data area. See “DISPLAY Statement” in Chapter 5.

A logical file opened for OUTPUT does not remove all records in the physical file on which it is based. The OVRDBF command can specify the first record to be made available to the program at execution time. See “OPEN Statement” in Chapter 5.

FIRST, PRIOR, and LAST can be specified on the READ statement for indexed files with dynamic access. See “READ Statement” in Chapter 5.

The KEY phrase of the START statement can specify EXTERNALLY-DESCRIBED-KEY. A comparison can be affected by the type of key fields in the record area defined for the file. See “START Statement” in Chapter 5.

For the WRITE statement, the mnemonic-name phrase can be used for stacker selection on a card punch file. See “WRITE Statement” in Chapter 5.

The composite of all operands in an arithmetic statement has a maximum of 30 digits. See “Arithmetic Statement Operands” in Chapter 5.

In the CORRESPONDING phrase of an arithmetic statement, the identifiers d1 and d2 can be subordinate to a FILLER item. See “Common Phrases, CORRESPONDING Phrase” in Chapter 5.

The two additional formats of the SET statement can be used to set mnemonic-names to on or off and to set condition-names to true. See “SET Statement” in Chapter 5.

Two active PERFORM and GO TO statements can have a common exit point. See “PERFORM Statement” in Chapter 5.

Input files do not need to be sequenced before a merge operation. See “SORT/MERGE” in Chapter 6.

In SORT/MERGE, input or output procedures can transfer control outside the input or output procedure using a PERFORM statement, or have control transferred inside the input or output procedure from elsewhere in the Procedure Division using a PERFORM statement, without causing compilation to fail. See “SORT/MERGE” in Chapter 6.

The COMMIT statement can be used to synchronize changes to records in data base files under commitment control, while preventing other jobs from accessing or modifying those records until the COMMIT is complete. See “COMMIT Statement” in Chapter 5.

The ROLLBACK statement can be used to cancel data base changes from files under commitment control when the changes should not remain permanent. See “ROLLBACK Statement” in Chapter 5.

COPY Statement – All Divisions

Pseudo-text for the REPLACING phrase of the COPY statement has considerations for division, section, and paragraph entries, and also for the placement of copied text as it appears in pseudo-text-2. See “COPY Statement, Replacing Phrase” in Chapter 6.

The file-name is optional. The default file-name is QCBLSRC. See “Qualification, Qualification Rules” in Chapter 2.

The COPY statement, DDS or DD format, is used to create Data Division entries for externally described files in a program. See “COPY Statement” in Chapter 6.

TRANSACTION Files

The data organization for work stations, display files, BSC files, communications files, and mixed files is TRANSACTION. TRANSACTION files have special formats for the file-control entry, file description entry, and the input/output statements. TRANSACTION file considerations are in Chapter 7.

Considerations for the TRANSACTION file-control entry include those for:

- The ASSIGN clause
- The ORGANIZATION clause
- The ACCESS MODE clause

- The FILE STATUS clause
- The CONTROL-AREA clause.

Considerations for the TRANSACTION file description entry are the same as those for other file description entries.

Boolean data provides a means of modifying and passing the values of the indicators associated with the display screen formats. See “Indicators” in Chapter 7.

The ACCEPT statement provides a way of accessing information about a program device when function-name is associated with a mnemonic-name of ATTRIBUTE-DATA in the SPECIAL-NAMES paragraph. See “ACCEPT Statement” in Chapter 7.

TRANSACTION file considerations for OPEN, CLOSE, READ, WRITE, REWRITE, and USE statements are given under the discussions for the respective statements and the discussions of the FORMAT, TERMINAL, INDICATORS, NO DATA, and SUBFILE phrases in Chapter 7.

The ACQUIRE statement can be used to acquire a program device for a TRANSACTION file. See “ACQUIRE Statement” in Chapter 7.

The DROP statement can be used to release a program device acquired by a TRANSACTION file. See “DROP Statement” in Chapter 7.

Compiler Options

Sequence checking can be suppressed at compile time. The apostrophe or the quotation mark can be used as a separator according to the compiler option specified. See “Create COBOL Program Command” and “PROCESS Statement” in Chapter 8.



Appendix C. Compile-Time Message Description

This appendix contains a general introduction to the compile-time messages. For additional information and a list of the messages, refer to the *Messages Guide: COBOL*. Compiler-generated messages indicate conditions encountered during program compilation. The messages describe an invalid use of COBOL syntax or a violation of system requirements.

System/38 COBOL provides the following message severity levels:

Severity	Meaning
00	Informational: This level is used to convey information to the user that may be of interest to him. No error has occurred. Informational messages are listed only when the FLAG (00) option is specified.
10	Warning: This level indicates that an error was detected but is not serious enough to interfere with the execution of the program.
20	Conditional: This level indicates that an error was made, but the compiler is taking a recovery that might yield the desired code.
30	Error: This level indicates that a serious error was detected. Compilation is completed, but execution of the program cannot be attempted.
40	Unrecoverable: This level indicates an error that forces termination of processing.

Note: 00, 10, and 20 messages are suppressed when the FLAG(30) option of the PROCESS statement is used or the CRTCLPGM command specifies FLAG(30) and is not overridden by the PROCESS statement. See "PROCESS Statement" in Chapter 8 for further information.

Compiler message numbers are assigned as follows:

Error Message	Description
CBL0000 through CBL0999	Messages with severity less than 30
CBL1000 through CBL1999	Messages with severity greater than 29
CBL8000 through CBL8999	FIPS messages

The compiler always attempts to provide full diagnostics of all source text in the program, even when errors have been detected. If the compiler cannot continue on a given statement, the message states that the compiler cannot continue and that it will ignore the rest of the statement. When this occurs, the programmer should examine the entire statement.

The System/38 message facility is used to produce all messages. The COBOL compiler messages reside in the message file, QCBLMSG.

Substitution variables and valid reply values are determined by the program sending the message, *not* by the message description stored in the message file. However, certain elements of a message description can be changed: for example, the text, severity level, default response, or dump list. To effect such changes, you need to define another message description using an Add Message Description (ADDMSGD) command, place the modified description in a user-created message file,¹ and specify that file in the Override Message File (OVRMSGF) command. Using the OVRMSGF command allows the compiler to retrieve messages from the specified file. See the ADDMSGD and OVRMSGF commands in the *CL Reference Manual* for additional information. The *CPF Programmer's Guide* describes the process of overriding message files.

CAUTION

Overriding an IBM-supplied message with a user-created message can produce results you do not anticipate. If reply values are not retained, the program might not respond to any replies. Changing default replies on *NOTIFY type messages could affect the ability of the program to run in unattended mode. Changing the severity could cancel a job not previously canceled. Be cautious when overriding IBM-supplied messages with user-created messages.

¹ If an IBM-supplied message must be changed and replaced in *its* message file, call your service representative.

Appendix D. Associated Card File Processing

The 5424 Multi-Function Card Unit (MFCU) can perform more than one card processing function in a single pass through the unit. If a card has already been partially punched, the MFCU can read the card, punch additional information into the card, and print up to 128 characters of information on the card. COBOL supports these combined functions through normal control language. This support is based on the concept of associated files.

COBOL handles each combined function as a separate logical file; each such logical file has its own file structure and processing requirements. Therefore, the user must define each function as if it were a unique file. However, because such combined function files refer to one physical unit, the user must define the logical files as being associated with each other, and relate them to each other during processing. The following sections explain the programming requirements for associated card file processing in System/38 COBOL.

Environment Division

Associated card file processing requires certain information in the SELECT and ASSIGN clauses.

SELECT Clause

A unique system file-name must be defined for each of the functions (reading, punching, and printing) to be combined.

ASSIGN Clause

For associated card files, the ASSIGN clause assignment-name must specify the primary (P) hopper of the 5424 MFCU and the association. The following format is valid:

$$\left. \begin{array}{l} \text{READER} \\ \text{PUNCH} \\ \text{PRINT} \\ \text{PUNCHPRINT} \end{array} \right\} [-\text{system-name}] - \text{P} - \text{association}$$

The association must be the same 1-digit integer for all associated logical files. This tells the compiler that each logical file is part of a particular associated card

file processing structure that is assigned to one physical unit. Any two, or all three, of the functions `READER`, `PUNCH`, or `PRINT` can each be specified once. When the function `PUNCHPRINT` is specified for an associated file, `READER` can be the only other function specified for that association.

More than one associated card file processing structure can be defined in a program; however, the structures must not be processed concurrently. Each such structure must have a unique association entry.

Data Division

An `FD` entry and a `01` record description entry must be defined in the File Section of the Data Division for each associated logical file.

Procedure Division

All associated files within one associated card file processing structure must be opened before a `READ` or `WRITE` statement is executed for any file in the structure. Similarly, no associated files can be closed until all `READ` and `WRITE` statements have been executed. When all such statements have been executed, all of the associated files must be closed.

An `OPEN`, `READ`, `WRITE`, or `CLOSE` statement for one of the associated files cannot be executed without concurrent processing of the other associated files. That is, all processing of associated files must reflect the association.

For associated files with the functions of read (`READER`), punch (`PUNCH`), print (`PRINT`), and punchprint (`PUNCHPRINT`), the following processing rules apply:

1. An `OPEN` statement must be executed for each file; with the `INPUT` phrase specified for the `READER` file and the `OUTPUT` phrase specified for the `PUNCH`, `PRINT`, and `PUNCHPRINT` files.
2. To make the logical input record available, a `READ` statement must first be executed for the `READER` file.
3. The next input/output operation executed for an associated file must be a `WRITE` statement for the associated `PUNCH` file. However, before the `WRITE` statement is executed, all data to be punched must be moved to the record specified for the associated `PUNCH` file.

Note: An alternative method is to move the additional data to be punched to the appropriate fields of the input record, and then to execute a `WRITE` statement for the `PUNCH` file using the `FROM` identifier phrase, where the identifier is the record-name specified for the `READER` file.

4. The `WRITE` statement for the `PRINT` file must be the next input/output statement executed for any of the associated files. Before the `WRITE` statement for the `PRINT` file is executed, you must format the data to be printed by moving it to the record specified for the `PRINT` file.

5. Steps 1 through 4 are repeated for all other records in the READER file.
6. When the PUNCHPRINT file is specified, a WRITE statement for the PUNCHPRINT file must be the next input/output statement following the READ statement for the READER file. Before the WRITE statement for the PUNCHPRINT file is executed, you must format the data to be punched and printed by moving it to the record specified for the PUNCHPRINT file.

When both PUNCH and PRINT files are associated, the stacker selection, if specified, in the WRITE statement for the PRINT file overrides any stacker selection specified for the PUNCH file.

When the READER file is specified for an associated card file, a READ until the AT END condition is detected must be done. This forces punch and/or print operations to occur for the last record read.

The order of processing specified in the preceding paragraphs also applies to a structure where only two of the three functions are associated, except that the processing of the unspecified function is not required.



Appendix E. Intermediate Result Fields

This appendix discusses the conceptual compiler algorithms for determining the number of integer and decimal places reserved for intermediate results. The following abbreviations are used:

- i Number of integer places carried for an intermediate result.
- d Number of decimal places carried for an intermediate result.
- dmax In a particular statement the larger of either:
 - The number of decimal places needed for the final result field(s)
 - The maximum number of decimal places defined for any operand except exponents and divisors.
- op1 First operand in a generated arithmetic statement.
- op2 Second operand in a generated arithmetic statement.
- d1,d2 Number of decimal places defined for op1 or op2, respectively.
- ir Intermediate result field obtained from the execution of a generated arithmetic statement or operation. Ir1, ir2, and so on represent successive intermediate results. Successive intermediate results may have the same location.

When an arithmetic statement contains only a single pair of operands, no intermediate results are generated. Intermediate results are possible in the following cases:

- In an ADD or SUBTRACT statement containing multiple operands immediately following the verb
- In a COMPUTE statement specifying a series of arithmetic operations
- In arithmetic expressions contained in an IF or PERFORM statement
- In the GIVING option with multiple result fields for the ADD, SUBTRACT, MULTIPLY, DIVIDE, or COMPUTE statements.

In such cases, the compiler treats the statement as a succession of operations. For example, the following statement:

COMPUTE Y = A + B * C - D / E + F ** G

is replaced by

F**G		yielding ir1
MULTIPLY B	BY C	yielding ir2
DIVIDE E	INTO D	yielding ir3
ADD A	TO ir2	yielding ir4
SUBTRACT ir3	FROM ir4	yielding ir5
ADD ir5	TO ir1	yielding Y

Compiler Calculation of Intermediate Results

The number of integer places in an ir is calculated as described in the following paragraphs:

The compiler first determines the maximum value that the ir can contain by performing the statement in which the ir occurs.

- If an operand in this statement is a data-name, the value used for the data-name is equal to the numerical value of the PICTURE for the data-name (that is, PICTURE 9V99 has the value 9.99).
- If an operand is a literal, the literal is treated as though it had a PICTURE, and the numerical value of the PICTURE is used (that is, the literal +127.3 has an implied PICTURE S999V9).
- If an operand is an intermediate result, the PICTURE determined for the intermediate result in a previous operation is used. The numerical value of that PICTURE is used.
- If the operation is division:
 - If op2 is a data-name, the value used for op2 is the minimum nonzero value of the digit in the PICTURE for the data-name (that is, PICTURE 9V99 has the value 0.01).
 - If op2 is an intermediate result, the intermediate result is treated as though it had a PICTURE, and the minimum nonzero value of the digits in this PICTURE is used.

When the maximum value of the ir is determined by the above procedures, i is set equal to the number of integers in the maximum value.

The number of decimal places contained in an ir is calculated as:

Operation	Decimal Places
+ or -	d1 or d2, whichever is greater
*	d1 + d2
/	d1 - d2 or dmax, whichever is greater

** dmax if op2 is nonintegral or a data-name; d1 * op2 if op2
is an integral literal

Note: The user must define the operands of any arithmetic statement with enough decimal places to give the desired accuracy in the final result.

Figure E-1 indicates the action of the compiler when handling intermediate results.

Value of $i + d$	Value of d	Value of $i + d_{max}$	Action Taken
< 30 = 30	Any value	Any value	i integer and d decimal places are carried for ir
> 30	< dmax = dmax	Any value	30 - d integer and d decimal places are carried for ir
	> dmax	< 30 = 30 > 30	i integer and 30 - i decimal places are carried for ir 30 - dmax integer and dmax decimal places are carried for ir

Figure E-1. Compiler Action on Intermediate Results



Appendix F. Sample File-Processing Programs

The programs in this appendix illustrate the fundamental programming techniques associated with each type of file organization. They are intended to be used for planning purposes only, and to illustrate the input/output statements necessary for certain access methods. Other COBOL features (the use of the ALTER statement and the PERFORM statement, for example) are used only incidentally. The programs are:

- Sequential File Creation
- Sequential File Updating and Extension
- Indexed File Creation
- Indexed File Updating
- Relative File Creation
- Relative File Updating
- Relative File Retrieval.

Sequential File Creation

This program creates a sequential file of employee salary records. The input records are arranged in ascending order of employee number. The output file has the identical order.

STMT SEQNBR -A 1 B.. ... 2 3 4 5 6 7 .IDENTFCN S COPYNAME CHG/DATE

```
1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID. CREATESEQ.
3 000300
4 000400 ENVIRONMENT DIVISION.
5 000500 CONFIGURATION SECTION.
6 000600 SOURCE-COMPUTER. IBM-S38.
7 000700 OBJECT-COMPUTER. IBM-S38.
8 000800 SPECIAL-NAMES. CONSOLE IS TYPEWRITER.
9 000900 INPUT-OUTPUT SECTION.
10 001000 FILE-CONTROL.
11 001100     SELECT INPUT-FILE ASSIGN TO DISK-FILEA
12 001200         FILE STATUS IS INPUT-FILE-STATUS.
13 001300     SELECT OUTPUT-FILE ASSIGN TO DISK-FILEB
14 001400         FILE STATUS IS OUTPUT-FILE-STATUS.
15 001500 DATA DIVISION.
16 001600 FILE SECTION.
17 001700 FD INPUT-FILE LABEL RECORDS STANDARD.
18 001800 01 INPUT-RECORD.
19 001900 05 INPUT-EMPLOYEE-NUMBER     PICTURE 9(6).
20 002000 05 INPUT-EMPLOYEE-NAME       PICTURE X(23).
21 002100 05 INPUT-EMPLOYEE-CODE       PICTURE 9.
22 002200 05 INPUT-EMPLOYEE-SALARY      PICTURE 9(6)V99.
23 002300 FD OUTPUT-FILE LABEL RECORDS STANDARD.
24 002400 01 OUTPUT-RECORD.
25 002500 05 OUTPUT-EMPLOYEE-NUMBER     PICTURE 9(6).
26 002600 05 OUTPUT-EMPLOYEE-NAME       PICTURE X(23).
27 002700 05 OUTPUT-EMPLOYEE-CODE       PICTURE 9.
28 002800 05 OUTPUT-EMPLOYEE-SALARY    PICTURE 9(6)V99.
29 002900 WORKING-STORAGE SECTION.
30 003000 77 INPUT-FILE-STATUS         PICTURE XX.
31 003100 77 OUTPUT-FILE-STATUS        PICTURE XX.
32 003200 01 INPUTEND                   PICTURE X VALUE SPACE.
33 003300 88 THE-END-OF-INPUT          VALUE "E".
34 003400 01 DISP-RECORD.
35 003500 05 OP-NAME                     PICTURE X(7).
36 003600 05 FILLER                       PICTURE XX VALUE SPACE.
37 003700 05 FILE-NAME                   PICTURE X(11).
38 003800 05 FILLER                       PICTURE XX VALUE SPACE.
39 003900 05 FILLER                       PICTURE X(14)
40 004000                               VALUE "FILE STATUS IS".
41 004100 05 FILLER                   PICTURE XX VALUE SPACE.
42 004200 05 SK                           PICTURE XX.
43 004300 PROCEDURE DIVISION.
004400 DECLARATIVES.
004500 I-O-ERROR SECTION.
004600     USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
004700                                     OUTPUT-FILE.
004800 I-O-ERROR-PARA.
004900*****
005000* DUMMY DECLARATIVES TO ENSURE CONTROL IS RETURNED TO THIS *
005100* PROGRAM WHEN AN ERROR OCCURS DURING FILE PROCESSING.     *
005200* ERROR HANDLING IS DONE AFTER EACH I/O STATEMENT.         *
005300*****
005400 END DECLARATIVES.
005500 MAIN-PROGRAM SECTION.
```

```

005600 OPEN-FILES.
44 005700 OPEN INPUT INPUT-FILE
005800 OUTPUT OUTPUT-FILE.
45 005900 IF INPUT-FILE-STATUS NOT = "00"
46 006000 MOVE "OPEN" TO OP-NAME
47 006100 MOVE "INPUT-FILE" TO FILE-NAME
48 006200 MOVE INPUT-FILE-STATUS TO SK
49 006300 PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2.
50 006400 IF OUTPUT-FILE-STATUS NOT = "00"
51 006500 MOVE "OPEN" TO OP-NAME
52 006600 MOVE "OUTPUT-FILE" TO FILE-NAME
53 006700 MOVE OUTPUT-FILE-STATUS TO SK
54 006800 PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2.
55 006900 PERFORM BUILD-FILE UNTIL THE-END-OF-INPUT.
007000 CLOSE-FILES.
56 007100 CLOSE INPUT-FILE
007200 OUTPUT-FILE.
57 007300 STOP RUN.
007400 BUILD-FILE.
58 007500 READ INPUT-FILE INTO OUTPUT-RECORD
59 007600 AT END SET THE-END-OF-INPUT TO TRUE.
60 007700 IF INPUT-FILE-STATUS NOT = "00"
61 007800 MOVE "READ" TO OP-NAME
62 007900 MOVE "INPUT-FILE" TO FILE-NAME
63 008000 MOVE INPUT-FILE-STATUS TO SK
64 008100 PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2
65 008200 GO TO CLOSE-FILES.
66 008300 WRITE OUTPUT-RECORD.
67 008400 IF OUTPUT-FILE-STATUS NOT = "00"
68 008500 MOVE "WRITE" TO OP-NAME
69 008600 MOVE "OUTPUT-FILE" TO FILE-NAME
70 008700 MOVE OUTPUT-FILE-STATUS TO SK
71 008800 PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2
72 008900 GO TO CLOSE-FILES.
009000 ERROR-OUT-1.
73 009100 DISPLAY "FILE PROCESSING ERROR" UPON TYPEWRITER.
74 009200 DISPLAY DISP-RECORD UPON TYPEWRITER.
75 009300 CLOSE INPUT-FILE
009400 OUTPUT-FILE.
76 009500 STOP RUN.
009600 ERROR-OUT-2.
009700 EXIT.

```

STMT SEQNBR MSGID SEV TEXT

* 44 005400 CBL0335 00 EMPTY PARAGRAPH OK SECTION PRECEDES *END DECLARATIVES* PARAGRAPH OR SECTION.

MESSAGE SUMMARY

TOTAL	INFO(0-4)	WARNING(5-19)	ERROR(20-29)	SEVERE(30-39)	TERMINAL(40-99)
1	1	0	0	0	0

97 SOURCE RECORDS READ
0 COPY RECORDS READ
0 COPY MEMBERS PROCESSED
0 SEQUENCE ERRORS
0 WAS THE HIGHEST SEVERITY MESSAGE ISSUED

Sequential File Updating and Extension

This program updates and extends the file created by the CREATE-SEQUENTIAL program. The INPUT-FILE and the I-O-FILE are each read. When a match is found between INPUT-EMPLOYEE-NUMBER and I-O-EMPLOYEE-NUMBER, the input record replaces the original record. After the I-O-FILE has been completely processed, new employee records are added at the end of the file.

```
STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S  COPYNAME  CHG/DATE

 1 000100 IDENTIFICATION DIVISION.
 2 000200 PROGRAM-ID. UPDATESEQ.
 3 000300 ENVIRONMENT DIVISION.
 4 000400 CONFIGURATION SECTION.
 5 000500 SOURCE-COMPUTER. IBM-S38.
 6 000600 OBJECT-COMPUTER. IBM-S38.
 7 000700 INPUT-OUTPUT SECTION.
 8 000800 FILE-CONTROL.
 9 000900     SELECT INPUT-FILE ASSIGN TO DISK-FILEB
10 001000     FILE STATUS IS INPUT-FILE-STATUS.
11 001100     SELECT MASTER-FILE ASSIGN TO DISK-MSTFILEB
12 001200     FILE STATUS IS MASTER-FILE-STATUS.
13 001300
14 001400 DATA DIVISION.
15 001500 FILE SECTION.
16 001600 FD INPUT-FILE LABEL RECORDS STANDARD.
17 001700 01 INPUT-RECORD.
18 001800     05 INPUT-EMPLOYEE-NUMBER     PICTURE 9(6).
19 001900     05 INPUT-EMPLOYEE-NAME       PICTURE X(28).
20 002000     05 INPUT-EMPLOYEE-CODE     PICTURE 9.
21 002100     05 INPUT-EMPLOYEE-SALARY     PICTURE 9(6)V99.
22 002200 FD MASTER-FILE LABEL RECORDS STANDARD.
23 002300 01 MASTER-RECORD.
24 002400     05 MST-EMPLOYEE-NUMBER       PICTURE 9(6).
25 002500     05 MST-EMPLOYEE-NAME       PICTURE X(28).
26 002600     05 MST-EMPLOYEE-CODE     PICTURE 9.
27 002700     05 MST-EMPLOYEE-SALARY     PICTURE 9(6)V99.
28 002800 WORKING-STORAGE SECTION.
29 002900 77 INPUT-FILE-STATUS           PICTURE XX.
30 003000 77 MASTER-FILE-STATUS     PICTURE XX.
31 003100 01 INPUTEND                   PICTURE X VALUE SPACE.
32 003200 88 THE-END-OF-INPUT           VALUE "E".
33 003300 01 MASTEREND                 PICTURE X VALUE SPACE.
34 003400 88 THE-END-OF-MASTER         VALUE "E".
35 003500 01 ERROR-INFO.
36 003600     05 OP-NAME                 PICTURE X(12).
37 003700     05 FILLER                  PICTURE XX VALUE SPACE.
38 003800     05 FILE-NAME               PICTURE X(11).
39 003900     05 FILLER                  PICTURE XX VALUE SPACE.
40 004000     05 FILLER                  PICTURE X(14)
41 004100     VALUE "FILE STATUS IS".
42 004200     05 FILLER                  PICTURE XX VALUE SPACE.
43 004300     05 SK                      PICTURE XX.
44 004400 PROCEDURE DIVISION.
45 004500 DECLARATIVES.
46 004600 INPUT-FILE-ERROR SECTION.
47 004700     USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
48 004800 INPUT-FILE-ERROR-PARA.
49 004900     MOVE INPUT-FILE-STATUS TO SK.
50 005000     MOVE "INPUT-FILE" TO FILE-NAME.
51 005100     DISPLAY "FILE PROCESSING ERROR".
52 005200     DISPLAY ERROR-INFO.
53 005300     DISPLAY "PROCESSING TERMINATED DUE TO I-O ERROR".
54 005400     STOP RUN.
55 005500 I-O-FILE-ERROR SECTION.
```

```

005600      USE AFTER STANDARD ERROR PROCEDURE ON MASTER-FILE.
005700 MASTER-FILE-ERROR-PARA.
51 005800      MOVE MASTER-FILE-STATUS TO SK.
52 005900      MOVE "MASTER-FILE" TO FILE-NAME.
53 006000      DISPLAY "FILE PROCESSING ERROR".
54 006100      DISPLAY ERROR-INFO.
55 006200      DISPLAY "PROCESSING TERMINATED DUE TO I-O ERROR".
56 006300      STOP RUN.
006400 END DECLARATIVES.
006500 MAIN-PROGRAM SECTION.
006600 OPEN-FILES.
57 006700      MOVE "OPEN" TO OP-NAME.
58 006800      OPEN INPUT INPUT-FILE
006900          I-O MASTER-FILE.
007000 PROCESSING-LOGIC.
59 007100      PERFORM READ-INPUT-FILE.
60 007200      PERFORM READ-MASTER-FILE.
61 007300      PERFORM PROCESS-FILES UNTIL THE-END-OF-INPUT.
007400 CLOSE-FILES.
62 007500      MOVE "CLOSE" TO OP-NAME.
63 007600      CLOSE MASTER-FILE
007700          INPUT-FILE.
64 007800      STOP RUN.
007900 READ-INPUT-FILE.
65 008000      MOVE "READ" TO OP-NAME.
66 008100      READ INPUT-FILE
67 008200          AT END SET THE-END-OF-INPUT TO TRUE.
008300 READ-MASTER-FILE.
68 008400      MOVE "READ" TO OP-NAME.
69 008500      READ MASTER-FILE
008600          AT END
70 008700          SET THE-END-OF-MASTER TO TRUE
71 008800          MOVE "AT END CLOSE" TO OP-NAME
72 008900          CLOSE MASTER-FILE
73 009000          MOVE "OPEN EXTEND" TO OP-NAME
74 009100          OPEN EXTEND MASTER-FILE.
009200 PROCESS-FILES.
75 009300      IF THE-END-OF-MASTER
76 009400          WRITE MASTER-RECORD FROM INPUT-RECORD
77 009500          PERFORM READ-INPUT-FILE
009600      ELSE
78 009700          IF MST-EMPLOYEE-NUMBER LESS THAN INPUT-EMPLOYEE-NUMBER
79 009800              PERFORM READ-MASTER-FILE
009900          ELSE
80 010000              IF MST-EMPLOYEE-NUMBER = INPUT-EMPLOYEE-NUMBER
81 010100                  MOVE "REWRITE" TO OP-NAME
82 010200                  REWRITE MASTER-RECORD FROM INPUT-RECORD
83 010300                  PERFORM READ-INPUT-FILE
84 010400                  PERFORM READ-MASTER-FILE
010500              ELSE
85 010600                  DISPLAY "ERROR RECORD -> ". INPUT-EMPLOYEE-NUMBER
86 010700                  PERFORM READ-INPUT-FILE.

```

TOTAL	INFO(0-4)	WARNING(5-19)	ERROR(20-29)	SEVERE(30-39)	TERMINAL(40-99)
0	0	0	0	0	0

```

107 SOURCE RECORDS READ
0 COPY RECORDS READ
0 COPY MEMBERS PROCESSED
0 SEQUENCE ERRORS
0 WAS THE HIGHEST SEVERITY MESSAGE ISSUED

```

Indexed File Creation

This program creates an indexed file of summary records for bank depositors. The key within each indexed file record is REC-ID (the depositor's account number); the input records are ordered in ascending sequence upon this key. Records are read from the input file and transferred to the indexed file record area. The indexed file record is then written.

STMT SEQNBR -A 1 B... 2 ... 3 ... 4 ... 5 ... 6 ... 7 .IDENTFCN S COPYNAME CHG/DATE

```
1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID. CREATEIND.
3 000300
4 000400 ENVIRONMENT DIVISION.
5 000500 CONFIGURATION SECTION.
6 000600 SOURCE-COMPUTER. IBM-S38.
7 000700 OBJECT-COMPUTER. IBM-S38.
8 000800 INPUT-OUTPUT SECTION.
9 000900 FILE-CONTROL.
10 001000 SELECT INDEXED-FILE ASSIGN TO DISK-INDEXFILE
11 001100 ORGANIZATION IS INDEXED
12 001200 ACCESS IS SEQUENTIAL
13 001300 RECORD KEY IS INDEX-KEY
14 001400 FILE STATUS IS INDEXED-FILE-STATUS.
15 001500 SELECT INPUT-FILE ASSIGN TO DISK-FILEG
16 001600 FILE STATUS IS INPUT-FILE-STATUS.
17 001700 DATA DIVISION.
18 001800 FILE SECTION.
19 001900 FD INDEXED-FILE LABEL RECORDS STANDARD.
20 002000 01 INDEX-RECORD.
21 002100 05 INDEX-KEY PICTURE X(10).
22 002200 05 INDEX-FLD1 PICTURE X(10).
23 002300 05 INDEX-NAME PICTURE X(20).
24 002400 05 INDEX-BAL PICTURE S9(5)V99.
25 002500 FD INPUT-FILE LABEL RECORDS STANDARD.
26 002600 01 INPUT-RECORD.
27 002700 05 INPUT-KEY PICTURE X(10).
28 002800 05 INPUT-NAME PICTURE X(20).
29 002900 05 INPUT-BAL PICTURE S9(5)V99.
30 003000 WORKING-STORAGE SECTION.
31 003100 77 INDEXED-FILE-STATUS PICTURE XX.
32 003200 77 INPUT-FILE-STATUS PICTURE XX.
33 003300 77 OP-NAME PICTURE X(7).
34 003400 01 INPUTEND PICTURE X VALUE SPACES.
35 003500 88 THE-END-OF-INPUT VALUE "E".
36 003600 01 ERRORFLAG PICTURE X VALUE SPACES.
37 003700 88 ERROR-OCCURRED VALUE "1".
38 003800 PROCEDURE DIVISION.
003900 DECLARATIVES.
004000 INPUT-ERROR SECTION.
004100 USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
004200 INPUT-ERROR-PARA.
39 004300 DISPLAY "UNEXPECTED ERROR ON ", OP-NAME, " FOR INPUT-FILE ".
40 004400 DISPLAY "FILE STATUS IS ", INPUT-FILE-STATUS.
41 004500 SET ERROR-OCCURRED TO TRUE.
004600 OUTPUT-ERROR SECTION.
004700 USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT.
004800 OUTPUT-ERROR-PARA.
42 004900 DISPLAY "UNEXPECTED ERROR ON ", OP-NAME, " FOR INDEXED-FILE"
43 005000 DISPLAY "FILE STATUS IS ", INDEXED-FILE-STATUS.
44 005100 SET ERROR-OCCURRED TO TRUE.
005200 END DECLARATIVES.
005300 MAIN-PROCESSING SECTION.
005400 MAIN-PROCEDURE.
45 005500 MOVE "OPEN" TO OP-NAME.
```

```

46 005600 OPEN INPUT INPUT-FILE
    005700 OUTPUT INDEXED-FILE.
47 005800 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
49 005900 PERFORM READ-INPUT-FILE.
50 006000 PERFORM LOAD-INDEXED-FILE THRU READ-INPUT-FILE
    006100 UNTIL THE-END-OF-INPUT.
51 006200 MOVE "CLOSE" TO OP-NAME.
52 006300 CLOSE INPUT-FILE
    006400 INDEXED-FILE.
53 006500 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
55 006600 STOP RUN.
    005700 LOAD-INDEXED-FILE.
56 006800 MOVE INPUT-KEY TO INDEX-KEY.
57 005900 MOVE INPUT-NAME TO INDEX-NAME.
58 007000 MOVE INPUT-BAL TO INDEX-BAL.
59 007100 MOVE SPACES TO INDEX-FLD1.
60 007200 MOVE "WRITE" TO OP-NAME.
61 007300 WRITE INDEX-RECORD
    007400 INVALID KEY
    007500 DISPLAY "WRITE FAILED FOR KEY ", INDEX-KEY.
63 007600 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
    007700 READ-INPUT-FILE.
65 007800 MOVE "READ" TO OP-NAME.
66 007900 READ INPUT-FILE
67 008000 AT END SET THE-END-OF-INPUT TO TRUE.
68 008100 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
    008200 ERROR-TERMINATION.
70 008300 DISPLAY "I-O ERROR OCCURRED - PROCESS TERMINATING".
71 008400 STOP RUN.

```

TOTAL	INFO(0-4)	WARNING(5-19)	ERROR(20-29)	SEVERE(30-39)	TERMINAL(40-99)
0	0	0	0	0	0

```

84 SOURCE RECORDS READ
0 COPY RECORDS READ
0 COPY MEMBERS PROCESSED
0 SEQUENCE ERRORS
0 WAS THE HIGHEST SEVERITY MESSAGE ISSUED

```


Indexed File Updating

This program, using dynamic access, updates the indexed file created in the CREATE-INDEXED program.

The input records contain the key for the record, the depositor name, and the amount of the transaction.

When the input record is read, the program tests whether this is a transaction record (in which case, all fields of the record are filled) or a record requesting sequential retrieval of a specific generic class (in which case, only the IN-GEN-FLD of the input record contains data).

Random access is used for the updating and printing of the transaction records. Sequential access is used for the retrieval and printing of all records within one generic class.

```
STMT SEQNBR -A 1 8.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S  COPYNAME  CHG/DATE

 1 000100 IDENTIFICATION DIVISION.
 2 000200 PROGRAM-ID. UPDATEIND.
 3 000300
 4 000400 ENVIRONMENT DIVISION.
 5 000500 CONFIGURATION SECTION.
 6 000600 SOURCE-COMPUTER. IBM-S38.
 7 000700 OBJECT-COMPUTER. IBM-S38.
 8 000800 INPUT-OUTPUT SECTION.
 9 000900 FILE-CONTROL.
10 001000     SELECT MASTER-FILE ASSIGN TO DISK-INDEXFILE
11 001100         ORGANIZATION IS INDEXED
12 001200         ACCESS IS DYNAMIC
13 001300         RECORD KEY IS MASTER-KEY
14 001400         FILE STATUS IS MASTER-FILE-STATUS.
15 001500     SELECT INPUT-FILE ASSIGN TO DISK-FILEM
16 001600         FILE STATUS IS INPUT-FILE-STATUS.
17 001700     SELECT PRINT-FILE ASSIGN TO PRINTER-OSYSRPT
18 001800         FILE STATUS IS PRINT-FILE-STATUS.
19 001900 DATA DIVISION.
20 002000 FILE SECTION.
21 002100 FD MASTER-FILE LABEL RECORDS STANDARD.
22 002200 01 MASTER-RECORD.
23 002300     05 MASTER-KEY.
24 002400         10 MASTER-GEN-FLD     PICTURE X(5).
25 002500         10 MASTER-DET-FLD     PICTURE X(5).
26 002600     05 MASTER-FLD1           PICTURE X(10).
27 002700     05 MASTER-NAME           PICTURE X(20).
28 002800     05 MASTER-BAL            PICTURE S9(5)V99.
29 002900 FD INPUT-FILE LABEL RECORDS STANDARD.
30 003000 01 INPUT-REC.
31 003100     05 INPUT-KEY.
32 003200         10 INPUT-GEN-FLD     PICTURE X(5).
33 003300         10 INPUT-DET-FLD     PICTURE X(5).
34 003400     05 INPUT-NAME            PICTURE X(20).
35 003500     05 INPUT-AMT             PICTURE S9(5)V99.
36 003600 FD PRINT-FILE LABEL RECORDS OMITTED
37 003700 LINAGE 12 LINES FOOTING AT 9.
38 003800 01 PRINT-RECORD-1.
39 003900     05 PRINT-KEY             PICTURE X(10).
40 004000     05 FILLER                 PICTURE X(5).
41 004100     05 PRINT-NAME            PICTURE X(20).
42 004200     05 FILLER                 PICTURE X(5).
43 004300     05 PRINT-BAL             PICTURE $$$$.$$$-99-.
44 004400     05 FILLER                 PICTURE X(5).
45 004500     05 PRINT-AMT             PICTURE $$$$.$$$-99-.
46 004600     05 FILLER                 PICTURE X(5).
47 004700     05 PRINT-NEW-BAL         PICTURE $$$$.$$$-99-.
48 004800 01 PRINT-RECORD-2.         PICTURE X(89).
49 004900 WORKING-STORAGE SECTION.
50 005000 77 MASTER-FILE-STATUS     PICTURE XX.
51 005100 77 INPUT-FILE-STATUS       PICTURE XX.
52 005200 77 PRINT-FILE-STATUS       PICTURE XX.
53 005300 01 PAGE-HEAD.
54 005400     05 FILLER                 PICTURE X(38) VALUE SPACES.
55 005500     05 FILLER                 PICTURE X(13) VALUE "UPDATE REPORT".
```

```

56 005600      05 FILLER          PICTURE X(38) VALUE SPACES.
57 005700 01  COLUMN-HEAD.
58 005800      05 FILLER          PICTURE X(6) VALUE "KEY ID".
59 005900      05 FILLER          PICTURE X(9) VALUE SPACES.
60 006000      05 FILLER          PICTURE X(4) VALUE "NAME".
61 006100      05 FILLER          PICTURE X(21) VALUE SPACES.
62 006200      05 FILLER          PICTURE X(11) VALUE "CUR BALANCE".
63 006300      05 FILLER          PICTURE X(5) VALUE SPACES.
64 006400      05 FILLER          PICTURE X(13) VALUE "UPDATE AMOUNT".
65 006500      05 FILLER          PICTURE X(4) VALUE SPACES.
66 006600      05 FILLER          PICTURE X(11) VALUE "NEW BALANCE".
67 006700      05 FILLER          PICTURE X(4) VALUE SPACES.
68 006800 01  PAGE-FOOT.
69 006900      05 FILLER          PICTURE X(91) VALUE SPACES.
70 007000      05 FILLER          PICTURE A(6) VALUE "PAGE ".
71 007100      05 PG-NUMBER       PICTURE 99 VALUE 00.
72 007200
73 007300 01  INPUTEND           PICTURE X VALUE SPACE.
74 007400      88 THE-END-OF-INPUT VALUE "E".
75 007500 01  ERRORFLAG         PICTURE X VALUE SPACE.
76 007600      88 ERROR-OCCURRED  VALUE "I".
77 007700 01  ERROR-DATA.
78 007800      05 FILLER          PICTURE X(21)
79 007900          VALUE "STATEMENT FAILING IS ".
80 009000      05 OP-NAME         PICTURE X(9).
81 009100      05 FILLER          PICTURE X(16) VALUE " FILE STATUS IS ".
82 008200      05 STATUS-VALUE    PICTURE XX.
83 008300 01  INPUT-MESSAGE.
84 008400      05 FILLER          PICTURE X(30)
85 008500          VALUE "UNEXPECTED ERROR ON INPUT-FILE" .
86 008600 01  I-O-MESSAGE.
87 008700      05 FILLER          PICTURE X(31)
88 008800          VALUE "UNEXPECTED ERROR ON MASTER-FILE" .
89 008900 01  OUTPUT-MESSAGE.
90 009000      05 FILLER          PICTURE X(30)
91 009100          VALUE "UNEXPECTED ERROR ON PRINT-FILE" .
92 009200 PROCEDURE DIVISION.
009300 DECLARATIVES.
009400 INPUT-ERROR SECTION.
009500 USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
009600 INPUT-ERROR-PARA.
93 009700 DISPLAY INPUT-MESSAGE.
94 009800 MOVE INPUT-FILE-STATUS TO STATUS-VALUE.
95 009900 DISPLAY ERROR-DATA.
96 010000 SET ERROR-OCCURRED TO TRUE.
010100 I-O-ERROR SECTION.
010200 USE AFTER STANDARD ERROR PROCEDURE ON I-O.
010300 I-O-ERROR-PARA.
97 010400 DISPLAY I-O-MESSAGE.
98 010500 MOVE MASTER-FILE-STATUS TO STATUS-VALUE.
99 010600 DISPLAY ERROR-DATA.
100 010700 SET ERROR-OCCURRED TO TRUE.
010800 OUTPUT-ERROR SECTION.
010900 USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT.
011000 OUTPUT-ERROR-PARA.
101 011100 DISPLAY OUTPUT-MESSAGE.
102 011200 MOVE PRINT-FILE-STATUS TO STATUS-VALUE.
103 011300 DISPLAY ERROR-DATA.
104 011400 SET ERROR-OCCURRED TO TRUE.
011500 END DECLARATIVES.
011600 MAIN-PROCESSING SECTION.
011700 MAIN-PROCEDURE.
105 011800 MOVE "OPEN" TO OP-NAME.
106 011900 OPEN INPUT INPUT-FILE
012000 I-O MASTER-FILE
012100 OUTPUT PRINT-FILE.
107 012200 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
109 012300 PERFORM PAGE-START.
110 012400 PERFORM READ-INPUT-FILE.
111 012500 PERFORM PROCESS-DATA THRU READ-INPUT-FILE
012600 UNTIL THE-END-OF-INPUT.
112 012700 MOVE "CLOSE" TO OP-NAME.
113 012800 CLOSE INPUT-FILE
012900 MASTER-FILE
013000 PRINT-FILE.
114 013100 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
116 013200 STOP RUN.
013300
013400 PROCESS-DATA.
117 013500 IF INPUT-DET-FLD EQUAL SPACES
118 013600 PERFORM INIT-SEQUENTIAL-PROCESS
013700 ELSE
119 013800 PERFORM DYNAMIC-PROCESS.
013900 READ-INPUT-FILE.
120 014000 MOVE "READ" TO OP-NAME.
121 014100 READ INPUT-FILE
122 014200 AT END SET THE-END-OF-INPUT TO TRUE.
123 014300 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
014400

```

```

014500 INIT-SEQUENTIAL-PROCESS.
125 014600 MOVE INPUT-GEN-FLD TO MASTER-GEN-FLD.
126 014700 MOVE "START" TO OP-NAME.
127 014800 START MASTER-FILE
      014900 KEY IS NOT LESS THAN MASTER-GEN-FLD
      015000 INVALID KEY
128 015100 DISPLAY "MASTER-FILE START FAILED: INVALID KEY ".
      015200 MASTER-GEN-FLD
129 015300 MOVE HIGH-VALUE TO MASTER-GEN-FLD.
130 015400 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
132 015500 PERFORM SEQUENTIAL-PROCESS
      015600 UNTIL INPUT-GEN-FLD NOT EQUAL MASTER-GEN-FLD.
      015700
      015800 SEQUENTIAL-PROCESS.
133 015900 MOVE "READ NEXT" TO OP-NAME.
134 015000 READ MASTER-FILE NEXT RECORD
135 016100 AT END MOVE HIGH-VALUE TO MASTER-GEN-FLD.
136 016200 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
138 016300 IF INPUT-GEN-FLD EQUAL MASTER-GEN-FLD
139 016400 MOVE MASTER-KEY TO PRINT-KEY
140 016500 MOVE MASTER-NAME TO PRINT-NAME
141 016600 MOVE MASTER-BAL TO PRINT-NEW-BAL
142 016700 PERFORM PRINT-DETAIL.
      015800
      016900 DYNAMIC-PROCESS.
143 017000 MOVE INPUT-KEY TO MASTER-KEY.
144 017100 MOVE "READ" TO OP-NAME.
145 017200 READ MASTER-FILE
      017300 INVALID KEY
146 017400 DISPLAY "MASTER-FILE READ FAILED: INVALID KEY ".
      017500 MASTER-KEY
147 017600 MOVE HIGH-VALUE TO MASTER-GEN-FLD.
148 017700 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
150 017800 IF INPUT-GEN-FLD EQUAL MASTER-GEN-FLD
151 017900 MOVE MASTER-KEY TO PRINT-KEY
152 019000 MOVE MASTER-NAME TO PRINT-NAME
153 018100 MOVE MASTER-BAL TO PRINT-BAL
154 018200 MOVE INPUT-AMT TO PRINT-AMT
155 018300 ADD INPUT-AMT TO MASTER-BAL
156 018400 MOVE MASTER-BAL TO PRINT-NEW-BAL
157 018500 PERFORM PRINT-DETAIL
158 018600 MOVE "REWRITE" TO OP-NAME
159 018700 REWRITE MASTER-RECORD
      018800 INVALID KEY
160 019900 DISPLAY "MASTER-FILE REWRITE FAILED: INVALID KEY ".
      019000 MASTER-KEY
161 019100 MOVE HIGH-VALUE TO MASTER-GEN-FLD.
162 019200 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
      019300 PRINT-DETAIL.
164 019400 MOVE "WRITE" TO OP-NAME.
165 019500 WRITE PRINT-RECORD-1
      019600 AT END-OF-PAGE
166 019700 PERFORM PAGE-END THROUGH PAGE-START.
167 019800 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
169 019900 MOVE SPACES TO PRINT-RECORD-1.
020000
020100 PAGE-END.
170 020200 MOVE "WRITE" TO OP-NAME.
171 020300 ADD 1 TO PG-NUMBER.
172 020400 WRITE PRINT-RECORD-2 FROM PAGE-FOOT
      020500 AFTER ADVANCING 3.
173 020600 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
      020700 PAGE-START.
175 020800 WRITE PRINT-RECORD-2 FROM PAGE-HEAD
      020900 AFTER ADVANCING PAGE.
176 021000 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
178 021100 MOVE SPACES TO PRINT-RECORD-2.
179 021200 WRITE PRINT-RECORD-2 FROM COLUMN-HEAD
      021300 AFTER ADVANCING 1 LINE.
180 021400 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
182 021500 MOVE SPACES TO PRINT-RECORD-2.
      021600 ERROR-TERMINATION.
183 021700 DISPLAY "PROCESS TERMINATING ABNORMALLY".
184 021800 STOP RUN.

```

TOTAL	INFO(0-4)	WARNING(5-19)	ERROR(20-29)	SEVERE(30-39)	TERMINAL(40-99)
0	0	0	0	0	0

```

218 SOURCE RECORDS READ
0 COPY RECORDS READ
0 COPY MEMBERS PROCESSED
0 SEQUENCE ERRORS
0 WAS THE HIGHEST SEVERITY MESSAGE ISSUED

```

Relative File Creation

This program creates a relative file of summary sales records using sequential access. Each record contains a five-year summary of unit and dollar sales for one week of the year; there are 51 records within the file, each representing one week.

Each input record represents the summary sales for one week of one year. The records for the first week of the last five years (in ascending order) are the first five input records. The records for the second week of the last five years are the next five input records, and so on. Thus, five input records fill one output record.

The RELATIVE KEY for the RELATIVE-FILE is not specified because it is not required for sequential access unless the START statement is used. (For updating, however, the key is INPUT-WEEK.)

```

STMT SEQNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S COPYNAME CHG/DATE
 1 000100 IDENTIFICATION DIVISION.
 2 000200 PROGRAM-ID. CREATEREL.
 3 000300
 4 000400 ENVIRONMENT DIVISION.
 5 000500 CONFIGURATION SECTION.
 6 000600 SOURCE-COMPUTER. IBM-S38.
 7 000700 OBJECT-COMPUTER. IBM-S38.
 8 000800 SPECIAL-NAMES. REQUESTOR IS REQUESTOR.
 9 000900 FILE-CONTROL.
10 001000     SELECT RELATIVE-FILE ASSIGN TO DISK-FILED
11 001100     ORGANIZATION IS RELATIVE
12 001200     ACCESS IS SEQUENTIAL
13 001300     FILE STATUS RELATIVE-FILE-STATUS.
14 001400     SELECT INPUT-FILE ASSIGN TO DISK-FILEC
15 001500     FILE STATUS INPUT-FILE-STATUS.
16 001600
17 001700 DATA DIVISION.
18 001800 FILE SECTION.
19 001900 FD RELATIVE-FILE LABEL RECORDS ARE STANDARD.
20 002000 01 RELATIVE-RECORD-01.
21 002100     05 RELATIVE-RECORD OCCURS 5 TIMES INDEXED BY REL-INDEX.
22 002200         10 RELATIVE-YEAR          PICTURE 99.
23 002300         10 RELATIVE-WEEK          PICTURE 99.
24 002400         10 RELATIVE-UNIT-SALES     PICTURE 59(6).
25 002500         10 RELATIVE-DOLLAR-SALES  PICTURE 59(9)V99.
26 002600 FD INPUT-FILE LABEL RECORDS STANDARD.
27 002700 01 INPUT-RECORD.
28 002800     05 INPUT-YEAR          PICTURE 99.
29 002900     05 INPUT-WEEK         PICTURE 99.
30 003000     05 INPUT-UNIT-SALES     PICTURE 59(6).
31 003100     05 INPUT-DOLLAR-SALES  PICTURE 59(9)V99.
32 003200 WORKING-STORAGE SECTION.
33 003300 77 INPUT-FILE-STATUS     PICTURE XX.
34 003400 77 RELATIVE-FILE-STATUS  PICTURE XX.
35 003500 01 WORK-RECORD.
36 003600     05 WORK-YEAR          PICTURE 99 VALUE 00.
37 003700     05 WORK-WEEK         PICTURE 99.
38 003800     05 WORK-UNIT-SALES     PICTURE 59(6).
39 003900     05 WORK-DOLLAR-SALES  PICTURE 59(9)V99.
40 004000 01 ERROR-INFO.
41 004100     05 OP-NAME           PICTURE X(5).
42 004200     05 FILLER           PICTURE X(10)
43 004300                         VALUE " ERROR ON ".
44 004400     05 FILE-NAME        PICTURE X(13).
45 004500     05 FILLER           PICTURE X(16)
46 004600                         VALUE " FILE STATUS IS ".
47 004700     05 STATUS-VALUE     PICTURE XX.
48 004800 01 ERROR-FLAG         PICTURE X VALUE SPACE.
49 004900     88 ERROR-OCCURRED  VALUE "1".

```

```

005600 I-O-ERROR SECTION.
005700     USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
005800 INPUT-FILE-ERROR.
55 005900     MOVE "INPUT-FILE" TO FILE-NAME.
56 006000     MOVE INPUT-FILE-STATUS TO STATUS-VALUE.
57 006100     SET ERROR-OCCURRED TO TRUE.
006200 RELATIVE-FILE-ERROR.
58 006300     MOVE "RELATIVE-FILE" TO FILE-NAME.
59 006400     MOVE RELATIVE-FILE-STATUS TO STATUS-VALUE.
60 006500     SET ERROR-OCCURRED TO TRUE.
006600 END DECLARATIVES.
006700 BEGIN-PROCESSING SECTION.
006800 PROCESSING-CONTROL.
61 006900     MOVE "OPEN" TO OP-NAME.
62 007000     OPEN INPUT INPUT-FILE
007100         OUTPUT RELATIVE-FILE.
63 007200     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
65 007300     SET REL-INDEX TO 1.
66 007400     PERFORM READ-INPUT-FILE.
67 007500     PERFORM PROCESS-DATA THRU READ-INPUT-FILE
007600         UNTIL THE-END-OF-INPUT.
68 007700     CLOSE RELATIVE-FILE INPUT-FILE.
69 007800     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
71 007900     STOP RUN.
008000 ERROR-TERMINATION.
72 008100     DISPLAY ERROR-INFO UPON REQUESTOR.
73 008200     DISPLAY "PROCESSING TERMINATED DUE TO I-O ERROR"
008300         UPON REQUESTOR.
74 008400     STOP RUN.
008500 PROCESS-DATA.
75 008600     MOVE INPUT-RECORD TO RELATIVE-RECORD (REL-INDEX).
76 008700     IF REL-INDEX NOT = 5
77 008800         SET REL-INDEX UP BY 1
008900     ELSE
78 009000         SET REL-INDEX TO 1
79 009100         PERFORM RELATIVE-FILE-WRITE.
009200 READ-INPUT-FILE.
80 009300     MOVE "READ" TO OP-NAME.
81 009400     READ INPUT-FILE
82 009500         AT END SET THE-END-OF-INPUT TO TRUE.
83 009600     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
009700 RELATIVE-FILE-WRITE.
85 009800     MOVE "WRITE" TO OP-NAME.
86 009900     WRITE RELATIVE-RECORD-01.
87 010000     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.

```

TOTAL	INFO(0-4)	WARNING(5-19)	ERROR(20-29)	SEVERE(30-39)	TERMINAL(40-99)
0	0	0	0	0	0

100 SOURCE RECORDS READ
0 COPY RECORDS READ
0 COPY MEMBERS PROCESSED
0 SEQUENCE ERRORS
0 WAS THE HIGHEST SEVERITY MESSAGE ISSUED

Relative File Updating

This program uses sequential access to update the file of summary sales records created in the CREATE-RELATIVE program. The updating program adds a record for the new year and deletes the oldest year's records from the RELATIVE-FILE.

The input record represents the summary sales record for one week of the preceding year. The RELATIVE KEY for the RELATIVE-FILE is present in the input record as INPUT-WEEK. The RELATIVE KEY is used to check that the record was correctly written.

STMT SEQNBR -A 1 B.. ... 2 3 4 5 6 7 .IDENTFCN S COPYNAME CHG/DATE

```
1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID. UPDATEREL.
3 000300
4 000400 ENVIRONMENT DIVISION.
5 000500 CONFIGURATION SECTION.
6 000600 SOURCE-COMPUTER. IBM-S38.
7 000700 OBJECT-COMPUTER. IBM-S38.
8 000800 INPUT-OUTPUT SECTION.
9 000900 FILE-CONTROL.
10 001000 SELECT RELATIVE-FILE ASSIGN TO DISK-FILED
11 001100 ORGANIZATION IS RELATIVE
12 001200 ACCESS IS SEQUENTIAL
13 001300 RELATIVE KEY INPUT-WEEK
14 001400 FILE STATUS RELATIVE-FILE-STATUS.
15 001500 SELECT INPUT-FILE ASSIGN TO DISK-FILEE
16 001600 FILE STATUS INPUT-FILE-STATUS.
17 001700
18 001800 DATA DIVISION.
19 001900 FILE SECTION.
20 002000 FD RELATIVE-FILE LABEL RECORDS STANDARD.
21 002100 01 RELATIVE-RECORD PICTURE X(105).
22 002200 FD INPUT-FILE LABEL RECORDS STANDARD.
23 002300 01 INPUT-RECORD.
24 002400 05 INPUT-YEAR PICTURE 99.
25 002500 05 INPUT-WEEK PICTURE 99.
26 002600 05 INPUT-UNIT-SALES PICTURE S9(6).
27 002700 05 INPUT-DOLLAR-SALES PICTURE S9(9)V99.
28 002800 WORKING-STORAGE SECTION.
29 002900 77 RELATIVE-FILE-STATUS PICTURE XX.
30 003000 77 INPUT-FILE-STATUS PICTURE XX.
31 003100 01 INPUT-END PICTURE X VALUE SPACE.
32 003200 88 THE-END-OF-INPUT VALUE "E".
33 003300 01 WORK-RECORD.
34 003400 05 FILLER PICTURE X(21).
35 003500 05 CURRENT-WORK-YEAR3 PICTURE X(84).
36 003600 05 NEW-WORK-YEAR.
37 003700 10 WORK-YEAR PICTURE 99.
38 003800 10 WORK-WEEK PICTURE 99.
39 003900 10 WORK-UNIT-SALES PICTURE S9(6).
40 004000 10 WORK-DOLLAR-SALES PICTURE S9(9)V99.
41 004100 66 WORK-OUT-RECORD RENAMES
42 004200 CURRENT-WORK-YEAR3 THROUGH NEW-WORK-YEAR.
43 004300 01 ERROR-MESSAGE.
44 004400 05 OP-NAME PICTURE X(7).
45 004500 05 FILLER PICTURE X(10)
46 004600 VALUE " ERROR ON ".
47 004700 05 FILE-NAME PICTURE X(13).
48 004800 05 FILLER PICTURE X(16)
49 004900 VALUE " FILE STATUS IS ".
50 005000 05 STATUS-VALUE PICTURE XX.
51 005100
52 005200 PROCEDURE DIVISION.
005300 DECLARATIVES.
005400 I-O-ERROR SECTION.
005500 USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE-FILE.
```

```

005600                                INPUT-FILE.
005700 ERROR-PROCEDURE.
53 005800     DISPLAY ERROR-MESSAGE.
54 005900     DISPLAY "PROCESSING TERMINATING".
55 006000     STOP RUN.
006100 END DECLARATIVES.
006200 MAIN-PROCEDURE SECTION.
006300 BEGIN-PROCESSING.
56 006400     MOVE "OPEN" TO OP-NAME.
57 006500     MOVE "INPUT-FILE" TO FILE-NAME.
58 006600     OPEN INPUT INPUT-FILE.
59 006700     MOVE "RELATIVE-FILE" TO FILE-NAME.
60 006800     OPEN I-O RELATIVE-FILE.
61 006900     PERFORM READ-FILES.
62 007000     PERFORM UPDATE-RELATIVE-FILE THRU READ-FILES
007100                                UNTIL THE-END-OF-INPUT.
63 007200     MOVE "CLOSE" TO OP-NAME.
64 007300     MOVE "INPUT-FILE" TO FILE-NAME.
65 007400     CLOSE INPUT-FILE.
66 007500     MOVE "RELATIVE-FILE" TO FILE-NAME.
67 007600     CLOSE RELATIVE-FILE.
68 007700     STOP RUN.
007800 UPDATE-RELATIVE-FILE.
69 007900     MOVE "REWRITE" TO OP-NAME.
70 008000     MOVE "RELATIVE-FILE" TO FILE-NAME.
71 008100     REWRITE RELATIVE-RECORD FROM WORK-OUT-RECORD.
008200 READ-FILES.
72 008300     MOVE "READ" TO OP-NAME.
73 008400     MOVE "RELATIVE-FILE" TO FILE-NAME.
74 008500     READ RELATIVE-FILE INTO WORK-RECORD
75 008600                                AT END SET THE-END-OF-INPUT TO TRUE.
76 008700     MOVE "INPUT-FILE" TO FILE-NAME.
77 008800     READ INPUT-FILE INTO NEW-WORK-YEAR
78 008900                                AT END SET THE-END-OF-INPUT TO TRUE.

```

TOTAL	INFO(0-4)	WARNING(5-9)	ERROR(20-29)	SEVERE(30-39)	TERMINAL(40-99)
0	0	0	0	0	0

```

89 SOURCE RECORDS READ
0 COPY RECORDS READ
0 COPY MEMBERS PROCESSED
0 SEQUENCE ERRORS
0 WAS THE HIGHEST SEVERITY MESSAGE ISSUED

```

Relative File Retrieval

This program, using dynamic access, retrieves the summary file created by the CREATE-RELATIVE program.

The records of the INPUT-FILE contain one required field (INPUT-WEEK), which is the RELATIVE KEY for RELATIVE-FILE, and one optional field (END-WEEK). An input record containing data in INPUT-WEEK and spaces in END-WEEK requests a printout for that one specific RELATIVE-RECORD; the record is retrieved through random access. An input record containing data in both INPUT-WEEK and END-WEEK requests a printout of all the RELATIVE-FILE records within the RELATIVE KEY range of INPUT-WEEK through END-WEEK, inclusive; these records are retrieved through sequential access.

```

STMT SEQNBR -A 1 B.. ... 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 .IDENTFCN S COPYNAME CHG/DATE

1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID. RETRVEREL.
3 000300
4 000400 ENVIRONMENT DIVISION.
5 000500 CONFIGURATION SECTION.
6 000600 SOURCE-COMPUTER. IBM-S38.
7 000700 OBJECT-COMPUTER. IBM-S38.
8 000800 SPECIAL-NAMES. REQUESTOR IS REQUESTOR.
9 000900 INPUT-OUTPUT SECTION.
10 001000 FILE-CONTROL.
11 001100 SELECT RELATIVE-FILE ASSIGN TO DISK-FILED
12 001200 ORGANIZATION IS RELATIVE
13 001300 ACCESS IS DYNAMIC
14 001400 RELATIVE KEY INPUT-WEEK
15 001500 FILE STATUS IS RELATIVE-FILE-STATUS.
16 001600 SELECT INPUT-FILE ASSIGN TO DISK-FILEF
17 001700 FILE STATUS IS INPUT-FILE-STATUS.
18 001800 SELECT PRINT-FILE ASSIGN TO PRINTER-QSYSPT
19 001900 FILE STATUS IS PRINT-FILE-STATUS.
20 002000
21 002100 DATA DIVISION.
22 002200 FILE SECTION.
23 002300 FD RELATIVE-FILE LABEL RECORDS STANDARD.
24 002400 01 RELATIVE-RECORD-01.
25 002500 05 RELATIVE-RECORD OCCURS 5 TIMES INDEXED BY REL-INDEX.
26 002600 10 RELATIVE-YEAR PICTURE 99.
27 002700 10 RELATIVE-WEEK PICTURE 99.
28 002800 10 RELATIVE-UNIT-SALES PICTURE S9(6).
29 002900 10 RELATIVE-DOLLAR-SALES PICTURE S9(9)V99.
30 003000 FD INPUT-FILE LABEL RECORDS STANDARD.
31 003100 01 INPUT-RECORD.
32 003200 05 INPUT-WEEK PICTURE 99.
33 003300 05 END-WEEK PICTURE 99.
34 003400 FD PRINT-FILE LABEL RECORDS OMITTED.
35 003500 01 PRINT-RECORD.
36 003600 05 PRINT-WEEK PICTURE 99.
37 003700 05 FILLER PICTURE X(5).
38 003800 05 PRINT-YEAR PICTURE 99.
39 003900 05 FILLER PICTURE X(5).
40 004000 05 PRINT-UNIT-SALES PICTURE ZZZ.ZZ9.
41 004100 05 FILLER PICTURE X(5).
42 004200 05 PRINT-DOLLAR-SALES PICTURE $$$$.$$$$.$$$$.99.
43 004300 WORKING-STORAGE SECTION.
44 004400 77 RELATIVE-FILE-STATUS PICTURE XX.
45 004500 77 INPUT-FILE-STATUS PICTURE XX.
46 004600 77 PRINT-FILE-STATUS PICTURE XX.
47 004700 77 HIGH-WEEK PICTURE 99 VALUE 53.
48 004800 77 OP-NAME PICTURE X(9).
49 004900 01 INPUTEND PICTURE X VALUE SPACE.
50 005000 88 THE-END-OF-INPUT VALUE "E".
51 005100 PROCEDURE DIVISION.
005200 DECLARATIVES.
005300 RELATIVE-FILE-ERROR SECTION.
005400 USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE-FILE.
005500 RELATIVE-ERROR-MSG.

```


STMT SEQNBR -A 1 B.. ... 2 3 4 5 6 7 .IDENTFCN S COPYNAME CHG/DATE

```

52 005600 DISPLAY OP-NAME, " ERROR ON RELATIVE-FILE ".
53 005700 DISPLAY "FILE STATUS VALUE IS ", RELATIVE-FILE-STATUS.
54 005800 DISPLAY "PROCESSING TERMINATED ".
55 005900 STOP RUN.
006000 INPUT-FILE-ERROR SECTION.
006100 USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
006200 INPUT-ERROR-MSG.
56 006300 DISPLAY OP-NAME, " ERPR ON INPUT-FILE ".
57 006400 DISPLAY "FILE STATUS VALUE IS ", INPUT-FILE-STATUS.
58 006500 DISPLAY "PROCESSING TERMINATED ".
59 006600 STOP RUN.
006700 PRINT-FILE-ERROR SECTION.
006800 USE AFTER STANDARD ERROR PROCEDURE ON PRINT-FILE.
006900 PRINT-ERROR-MSG.
60 007000 DISPLAY OP-NAME, " ERROR ON PRINT-FILE ".
61 007100 DISPLAY "FILE STATUS VALUE IS ", PRINT-FILE-STATUS.
62 007200 DISPLAY "PROCESSING TERMINATED ".
63 007300 STOP RUN.
007400 END DECLARATIVES.
007500 MAIN-PROCEDURE SECTION.
007600 MAIN-PROCESSING.
64 007700 MOVE "OPEN" TO OP-NAME.
65 007800 OPEN INPUT INPUT-FILE RELATIVE-FILE
007900 OUTPUT PRINT-FILE.
66 008000 MOVE SPACES TO PRINT-RECORD.
67 008100 PERFORM READ-INPUT-FILE.
68 008200 PERFORM CONTROL-PROCESS THRU READ-INPUT-FILE
008300 UNTIL THE-END-OF-INPUT.
69 008400 MOVE "CLOSE" TO OP-NAME.
70 008500 CLOSE RELATIVE-FILE
008600 INPUT-FILE
008700 PRINT-FILE.
71 008800 STOP RUN.
008900 CONTROL-PROCESS.
72 009000 IF (END-WEEK = SPACES OR END-WEEK = 00)
73 009100 PERFORM RANDOM-PROCESS
009200 ELSE
74 009300 PERFORM SEQUENTIAL-PROCESS.
009400 READ-INPUT-FILE.
75 009500 MOVE "READ" TO OP-NAME.
76 009600 READ INPUT-FILE
77 009700 AT END SET THE-END-OF-INPUT TO TRUE.
009800 RANDOM-PROCESS.
78 009900 MOVE "READ" TO OP-NAME.
79 010000 READ RELATIVE-FILE
80 010100 INVALID KEY MOVE HIGH-WEEK TO RELATIVE-WEEK(1).
81 010200 IF RELATIVE-WEEK(1) NOT EQUAL HIGH-WEEK
82 010300 PERFORM PRINT-SUMMARY VARYING REL-INDEX FROM 1 BY 1
010400 UNTIL REL-INDEX > 5.
010500 SEQUENTIAL-PROCESS.
83 010600 MOVE "READ" TO OP-NAME.
84 010700 READ RELATIVE-FILE
85 010800 INVALID KEY MOVE HIGH-WEEK TO RELATIVE-WEEK(1).
86 010900 PERFORM READ-REL-SEQ
011000 UNTIL RELATIVE-WEEK(1) GREATER THAN END-WEEK.

```

STMT SEQNBR -A 1 B.. ... 2 3 4 5 6 7 .IDENTFCN S COPYNAME CHG/DATE

```

011100
011200 READ-REL-SEQ.
87 011300 PERFORM PRINT-SUMMARY VARYING REL-INDEX FROM 1 BY 1
011400 UNTIL REL-INDEX > 5.
88 011500 MOVE "READ NEXT" TO OP-NAME.
89 011600 READ RELATIVE-FILE NEXT RECORD
90 011700 AT END MOVE HIGH-WEEK TO RELATIVE-WEEK(1).
011800 PRINT-SUMMARY.
91 011900 MOVE RELATIVE-YEAR (REL-INDEX) TO PRINT-YEAR.
92 012000 MOVE RELATIVE-WEEK (REL-INDEX) TO PRINT-WEEK.
93 012100 MOVE RELATIVE-UNIT-SALES (REL-INDEX) TO PRINT-UNIT-SALES.
94 012200 MOVE RELATIVE-DOLLAR-SALES (REL-INDEX) TO PRINT-DOLLAR-SALES.
95 012300 MOVE "WRITE" TO OP-NAME.
96 012400 WRITE PRINT-RECORD AFTER ADVANCING 2 LINES.

```

TOTAL	INFO(0-4)	WARNING(5-19)	ERROR(20-29)	SEVERE(30-39)	TERMINAL(40-99)
0	0	0	0	0	0

```

124 SOURCE RECORDS READ
0 COPY RECORDS READ
0 COPY MEMBERS PROCESSED
0 SEQUENCE ERRORS
0 WAS THE HIGHEST SEVERITY MESSAGE ISSUED

```

Appendix G. COBOL Reserved Words

The following COBOL reserved word list identifies all reserved words in:

- IBM System/38 COBOL
- American National Standard COBOL, X3.23-1974
- CODASYL COBOL (from *CODASYL COBOL Journal of Development* dated December 1978).

Each word in the list is preceded by an identifier that is associated with one of the following meanings:

- | | |
|-------|--|
| Blank | A System/38 COBOL reserved word from the 1974 ANS standard. |
| 1 | A System/38 COBOL reserved word that is an IBM extension to the 1974 ANS standard. |
| 2 | A COBOL reserved word from the 1974 ANS standard that is not used by System/38 COBOL. These words should not be used if compatibility is important to an installation. If used, a diagnostic message will be issued. |
| 3 | A CODASYL COBOL reserved word that is not included in the 1974 ANS standard and is not supported by System/38 COBOL as an extension. If used, a diagnostic message will be issued. These words are included for compatibility. |

ACCEPT
 ACCESS
 1 ACQUIRE
 ADD
 ADVANCING
 AFTER
 ALL
 3 ALPHABET
 ALPHABETIC
 3 ALPHANUMERIC
 3 ALPHANUMERIC-EDITED
 ALSO
 ALTER
 ALTERNATE
 AND
 3 ANY
 ARE
 AREA
 AREAS
 ASCENDING
 ASSIGN
 AT
 1 ATTRIBUTE
 AUTHOR

 BEFORE
 3 BEGINNING
 3 BINARY
 3 BIT
 3 BITS
 BLANK
 BLOCK
 3 BOOLEAN
 BOTTOM
 BY

 CALL
 CANCEL
 2 CD
 2 CF
 2 CH
 CHARACTER
 CHARACTERS
 2 CLOCK-UNITS
 CLOSE
 2 COBOL
 2 CODE
 CODE-SET
 COLLATING
 2 COLUMN
 1 COMMA
 1 COMMIT
 1 COMMITMENT

3 COMMON
 2 COMMUNICATION
 COMP
 1 COMP-3
 1 COMP-4
 COMPUTATIONAL
 1 COMPUTATIONAL-3
 1 COMPUTATIONAL-4
 COMPUTE
 CONFIGURATION
 CONNECT
 CONTAINS
 3 CONTENT
 3 CONTINUE
 1 CONTROL
 1 CONTROL-AREA
 2 CONTROLS
 3 CONVERSION
 3 CONVERTING
 COPY
 CORR
 CORRESPONDING
 COUNT
 CURRENCY
 3 CURRENT

 DATA
 DATE
 DATE-COMPILED
 DATE-WRITTEN
 DAY
 3 DAY-OF-WEEK
 3 DB
 3 DB-ACCESS-CONTROL-KEY
 3 DB-DATA-NAME
 3 DB-EXCEPTION
 1 DB-FORMAT-NAME
 3 DB-RECORD-NAME
 3 DB-SET-NAME
 3 DB-STATUS
 2 DE
 DEBUG-CONTENTS
 DEBUG-ITEM
 3 DEBUG-LENGTH
 DEBUG-LINE
 DEBUG-NAME
 3 DEBUG-NUMERIC-CONTENTS
 3 DEBUG-SIZE
 3 DEBUG-START
 3 DEBUG-SUB
 DEBUG-SUB-1
 DEBUG-SUB-2
 DEBUG-SUB-3

3 DEBUG-SUB-ITEM
 3 DEBUG-SUB-N
 3 DEBUG-SUB-NUM
 DEBUGGING
 DECIMAL-POINT
 DECLARATIVES
 DELETE
 DELIMITED
 DELIMITER
 DEPENDING
 DESCENDING
 2 DESTINATION
 2 DETAIL
 2 DISABLE
 3 DISCONNECT
 DISPLAY
 3 DISPLAY-n
 DIVIDE
 DIVISION
 DOWN
 1 DROP
 3 DUPLICATE
 DUPLICATES
 DYNAMIC

 2 EGI
 ELSE
 2 EMI
 3 EMPTY
 2 ENABLE
 END
 3 END-ADD
 3 END-CALL
 3 END-COMPUTE
 3 END-DELETE
 3 END-DIVIDE
 3 END-EVALUATE
 3 END-IF
 3 END-MULTIPLY
 END-OF-PAGE
 3 END-PERFORM
 3 END-READ
 3 END-RECEIVE
 3 END-RETURN
 3 END-REWRITE
 3 END-SEARCH
 3 END-START
 3 END-STRING
 3 END-SUBTRACT
 3 END-UNSTRING
 3 END-WRITE

3	ENDING		IN		3	MODIFY
	ENTER		INDEX			MODULES
	ENVIRONMENT	3	INDEX-n			MOVE
	EOP		INDEXED			MULTIPLE
	EQUAL	1	INDIC			MULTIPLY
3	EQUALS	2	INDICATE			
3	ERASE	1	INDICATOR			NATIVE
	ERROR	1	INDICATORS			NEGATIVE
2	ESI		INITIAL			NEXT
3	EVALUATE	3	INITIALIZE			NO
	EVERY	2	INITIATE		3	NON-NULL
3	EXCEEDS		INPUT			NOT
	EXCEPTION		INPUT-OUTPUT		3	NULL
3	EXCLUSIVE		INSPECT		2	NUMBER
	EXIT		INSTALLATION			NUMERIC
3	EXOR		INTO		3	NUMERIC-EDITED
	EXTEND		INVALID			
3	EXTERNAL		IS			OBJECT-COMPUTER
1	EXTERNALLY-DESCRIBED-KEY		JUST			OCCURS
			JUSTIFIED			OF
3	FALSE					OFF
	FD					OMITTED
	FILE	3	KEEP			ON
	FILE-CONTROL		KEY			OPEN
	FILLER					OPTIONAL
2	FINAL		LABEL			OR
3	FIND		LAST		3	ORDER
3	FINISH	3	LD			ORGANIZATION
	FIRST		LEADING		3	OTHER
	FOOTING		LEFT			OUTPUT
	FOR	2	LENGTH			OVERFLOW
1	FORMAT		LESS		3	OWNER
3	FREE	2	LIMIT			
	FROM	2	LIMITS		3	PACKED-DECIMAL
			LINAGE		3	PADDING
2	GENERATE		LINAGE-COUNTER			PAGE
3	GET		LINE		2	PAGE-COUNTER
	GIVING	2	LINE-COUNTER			PERFORM
3	GLOBAL		LINES		2	PF
	GO		LINKAGE		2	PH
	GREATER	3	LOCALLY			PIC
2	GROUP		LOCK			PICTURE
			LOW-VALUE		2	PLUS
2	HEADING		LOW-VALUES			POINTER
	HIGH-VALUE					POSITION
	HIGH-VALUES	3	MEMBER			POSITIVE
			MEMORY		2	PRINTING
	I-O		MERGE		1	PRIOR
	I-O-CONTROL	2	MESSAGE			PROCEDURE
	IDENTIFICATION		MODE			PROCEDURES
	IF	1	MODIFIED			PROCEED

1 PROCESS
 PROGRAM
 PROGRAM-ID
 3 PROTECTED
 3 PURGE

 2 QUEUE
 QUOTE
 QUOTES

 RANDOM
 2 RD
 READ
 3 READY
 3 REALM
 3 REALMS
 2 RECEIVE
 3 RECONNECT
 RECORD
 3 RECORD-NAME
 RECORDS
 REDEFINES
 REEL
 3 REFERENCE
 3 REFERENCE-MODIFIER
 REFERENCES
 RELATIVE
 RELEASE
 REMAINDER
 REMOVAL
 RENAMES
 3 REPLACE
 REPLACING
 2 REPORT
 2 REPORTING
 2 REPORTS
 RERUN
 RESERVE
 2 RESET
 3 RETAINING
 3 RETRIEVAL
 RETURN
 REVERSED
 REWIND
 REWRITE
 2 RD
 2 RH
 RIGHT
 1 ROLLBACK
 1 ROLLING
 ROUNDED
 RUN

SAME
 SD
 SEARCH
 SECTION
 SECURITY
 2 SEGMENT
 SEGMENT-LIMIT
 SELECT
 2 SEND
 SENTENCE
 SEPARATE
 SEQUENCE
 SEQUENTIAL
 SET
 3 SETS
 SIGN
 SIZE
 SORT
 SORT-MERGE
 2 SOURCE
 SOURCE-COMPUTER
 SPACE
 SPACES
 SPECIAL-NAMES
 STANDARD
 STANDARD-1
 3 STANDARD-2
 START
 1 STARTING
 STATUS
 STOP
 3 STORE
 STRING
 2 SUB-QUEUE-1
 2 SUB-QUEUE-2
 2 SUB-QUEUE-3
 3 SUB-SCHEMA
 1 SUBFILE
 SUBTRACT
 2 SUM
 2 SUPPRESS
 2 SYMBOLIC
 SYNC
 SYNCHRONIZED

 2 TABLE
 TALLYING
 TAPE
 3 TENANT

TERMINAL
 2 TERMINATE
 3 TEST
 2 TEXT
 THAN
 1 THEN
 THROUGH
 THRU
 TIME
 TIMES
 TO
 TOP
 TRAILING
 1 TRANSACTION
 1 TRUE
 2 TYPE

 UNIT
 UNSTRING
 UNTIL
 UP
 3 UPDATE
 UPON
 USAGE
 3 USAGE-MODE
 USE
 USING

 VALUE
 VALUES
 VARYING

 WHEN
 WITH
 3 WITHIN
 WORDS
 WORKING-STORAGE
 WRITE

 ZERO
 ZEROES
 ZEROS

 +
 -
 *
 /
 **
 >
 <
 =

Appendix H. EBCDIC and ASCII Collating Sequences

The ascending collating sequences for both the EBCDIC (Extended Binary Coded Decimal Interchange Code) and ASCII (American National Standard Code for Information Interchange) character sets are given in this appendix. Decimal positions within the sequence are given, as well as the binary representation, symbol, meaning for each character, and corresponding decimal position within the other sequence. The symbols with an asterisk * in the righthand column do not correspond to the same symbol in the other collating sequence.

Note: When using the literal option of the alphabet-name clause, 1 must be added to the number shown in this appendix to specify the corresponding character. (The numbers in this appendix run from 0 to 255; the numbers in the literal option run from 1 to 256.)

EBCDIC Collating Sequence

Collating Sequence	Binary Representation	Symbol	Meaning	ASCII Number
0	00000000			0
64	01000000	SP	Space	32
74	01001010	¢	Cent sign	91 *
75	01001011	.	Period, decimal point	46
76	01001100	<	Less-than sign	60
77	01001101	(Left parenthesis	40
78	01001110	+	Plus sign	43
79	01001111		Vertical bar, logical OR	33 *
80	01010000	&	Ampersand	38
90	01011010	!	Exclamation point	93 *
91	01011011	\$	Dollar sign	36
92	01011100	*	Asterisk	42
93	01011101)	Right parenthesis	41
94	01011110	;	Semicolon	59
95	01011111	¬	Logical NOT	94
96	01100000	-	Minus, hyphen	45
97	01100001	/	Slash	47
106	01101010		Broken vertical bar	124 *
107	01101011	,	Comma	44
108	01101100	%	Percent sign	37
109	01101101	_	Underscore	95
110	01101110	>	Greater-than sign	62
111	01101111	?	Question mark	63
121	01111001	`	Accent grave	96
122	01111010	:	Colon	58
123	01111011	#	Number sign	35
124	01111100	@	At sign	64
125	01111101	'	Apostrophe, prime	39
126	01111110	=	Equal sign	61
127	01111111	"	Quotation mark	34
129	10000001	a		97
130	10000010	b		98
131	10000011	c		99
132	10000100	d		100
133	10000101	e		101
134	10000110	f		102
135	10000111	g		103
136	10001000	h		104
137	10001001	i		105

Collating Sequence	Binary Representation	Symbol	Meaning	ASCII Number
.				
145	10010001	j		106
146	10010010	k		107
147	10010011	l		108
148	10010100	m		109
149	10010101	n		110
150	10010110	o		111
151	10010111	p		112
152	10011000	q		113
153	10011001	r		114
.				
161	10100001	~	Tilde	126
162	10100010	s		115
163	10100011	t		116
164	10100100	u		117
165	10100101	v		118
166	10100110	w		119
167	10100111	x		120
168	10101000	y		121
169	10101001	z		122
.				
192	11000000	{	Left brace	123
193	11000001	A		65
194	11000010	B		66
195	11000011	C		67
196	11000100	D		68
197	11000101	E		69
198	11000110	F		70
199	11000111	G		71
200	11001000	H		72
201	11001001	I		73
.				
208	11010000	}	Right brace	125
209	11010001	J		74
210	11010010	K		75
211	11010011	L		76
212	11010100	M		77
213	11010101	N		78
214	11010110	O		79
215	11010111	P		80
216	11011000	Q		81
217	11011001	R		82
.				
224	11100000	\	Reverse slant	92
.				
226	11100010	S		83
227	11100011	T		84
228	11100100	U		85
229	11100101	V		86
230	11100110	W		87
231	11100111	X		88
232	11101000	Y		89

Collating Sequence	Binary Representation	Symbol	Meaning	ASCII Number
233	11101001	Z		90
.				
240	11110000	0		48
241	11110001	1		49
242	11110010	2		50
243	11110011	3		51
244	11110100	4		52
245	11110101	5		53
246	11110110	6		54
247	11110111	7		55
248	11111000	8		56
249	11111001	9		57
.				
.				
.				
255				

ASCII Collating Sequence

Collating Sequence	Binary Representation	Symbol	Meaning	EBCDIC Number
0	00000000		Null	0
.				
32	00100000	SP	Space	64
33	00100001	!	Exclamation point	79 *
34	00100010	"	Quotation mark	127
35	00100011	#	Number sign	123
36	00100100	\$	Dollar sign	91
37	00100101	%	Percent	108
38	00100110	&	Ampersand	80
39	00100111	'	Apostrophe, prime	125
40	00101000	(Opening parenthesis	77
41	00101001)	Closing parenthesis	93
42	00101010	*	Asterisk	92
43	00101011	+	Plus	78
44	00101100	,	Comma	107
45	00101101	-	Hyphen, minus	96
46	00101110	.	Period, decimal point	75
47	00101111	/	Slant	97
48	00110000	0		240
49	00110001	1		241
50	00110010	2		242
51	00110011	3		243
52	00110100	4		244
53	00110101	5		245
54	00110110	6		246
55	00110111	7		247
56	00111000	8		248
57	00111001	9		249
58	00111010	:	Colon	122
59	00111011	;	Semicolon	94
60	00111100	<	Less-than sign	76
61	00111101	=	Equals	126
62	00111110	>	Greater-than sign	110
63	00111111	?	Question mark	111
64	01000000	@	At sign	124
65	01000001	A		193
66	01000010	B		194
67	01000011	C		195
68	01000100	D		196
69	01000101	E		197
70	01000110	F		198
71	01000111	G		199
72	01001000	H		200
73	01001001	I		201
74	01001010	J		209
75	01001011	K		210
76	01001100	L		211
77	01001101	M		212

Collating Sequence	Binary Representation	Symbol	Meaning	EBCDIC Number
78	01001110	N		213
79	01001111	O		214
80	01010000	P		215
81	01010001	Q		216
82	01010010	R		217
83	01010011	S		226
84	01010100	T		227
85	01010101	U		228
86	01010110	V		229
87	01010111	W		230
88	01011000	X		231
89	01011001	Y		232
90	01011010	Z		233
91	01011011	[Opening bracket	74 *
92	01011100	\	Reverse slant	224
93	01011101]	Closing bracket	90 *
94	01011110	^	Circumflex, Logical NOT	95
95	01011111	⏟	Underscore	109
96	01100000	˘	Grave accent	121
97	01100001	a		129
98	01100010	b		130
99	01100011	c		131
100	01100100	d		132
101	01100101	e		133
102	01100110	f		134
103	01100111	g		135
104	01101000	h		136
105	01101001	i		137
106	01101010	j		145
107	01101011	k		146
108	01101100	l		147
109	01101101	m		148
110	01101110	n		149
111	01101111	o		150
112	01110000	p		151
113	01110001	q		152
114	01110010	r		153
115	01110011	s		162
116	01110100	t		163
117	01110101	u		164
118	01110110	v		165
119	01110111	w		166
120	01111000	x		167
121	01111001	y		168
122	01111010	z		169
123	01111011	{	Opening brace	192
124	01111100		Vertical line	106 *
125	01111101	}	Closing brace	208
126	01111110	~	Tilde	161

Appendix I. File Structure Support Summary and Status Key Values

Figure I-1 lists the required and optional entries for various types of file structures supported. Any file with a device type of disk can be assigned to a data base or non-data base auxiliary storage file. The codes used are as follows:

- Not applicable
- B Optional for a work station that supports subfiles
- C Optional entry, treated as comments only
- D Optional for file assigned to -DATABASE, not allowed if not assigned to a data base file
- I Optional for a file opened for input or input-output
- O Optional
- R Required
- S Required for a work station that supports subfiles
- X Required; syntax-checked, but treated as documentation.

Figure I-2 contains status key values and their meanings.

DEVICE TYPE	CARD READER	CARD PUNCH	CARD PRINT	CARD PUNCHPRINT	PRINTER	TAPE	DISK SEQ.	DISK REL SEQ.	DISK REL RANDOM	DISK REL DYNAMIC	DISK IDX SEQ.	DISK IDX RANDOM	DISK IDX DYNAMIC	WORK STATION	DISKETTE	FORMATFILE
ENVIRONMENT DIVISION																
RERUN ... RECORDS	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
SAME	O	O	O	O	O	O	O	.	O	O	O	O	O	O	O	O
AREA	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
RECORD AREA	O	O	O	O	O	O	O	.	O	O	O	O	O	O	O	O
SORT AREA	C	C	C
SORT MERGE AREA	C	C	C
MULTIPLE FILE TAPE	C
COMMITMENT CONTROL	D	D	D	D	D	D	D	.	.	.
SELECT	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
ASSIGN	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
OPTIONAL	I	I	I	I	I
ORGANIZATION	O	O	O	O	O	O	O	R	R	R	R	R	R	R	O	O
SEQUENTIAL	O	O	O	O	O	O	O	O	O
RELATIVE	R	R	R
INDEXED	R	R	R	.	.	.
TRANSACTION	R	.	.
ACCESS	O	O	O	O	O	O	O	O	R	R	O	R	R	O	O	O
SEQUENTIAL	O	O	O	O	O	O	O	O	.	.	O	.	.	O	O	O
RANDOM	R	.	.	R
DYNAMIC	R	.	.	R	S	.	.
RESERVE	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
RELATIVE KEY	O	R	R	.	.	.	S	.	.
RECORD KEY	R	R	R	.	.	.
DUPLICATES	D	D	D	.	.	.
FILE STATUS	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
CONTROL-AREA	O	.	.
DATA DIVISION																
LABEL RECORDS	X	X	X	X	X	R	X	X	X	X	X	X	X	X	X	X
STANDARD	O	R	R	R	R	R	R	R	O	R	R
OMITTED	R	R	R	R	R	O	O	.	.
VALUE OF	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
OF																
BLOCK CONTAINS	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
RECORD CONTAINS	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
DATA RECORDS	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
CODE-SET	O	O	.
LINAGE	O

Figure I-1 (Part 1 of 3). File Structure Support

DEVICE TYPE	CARD READER	CARD PUNCH	CARD PRINT	CARD PUNCHPRINT	PRINTER	TAPE	DISK SEQ.	DISK REL SEQ	DISK REL RANDOM	DISK REL DYNAMIC	DISK IDX SEQ	DISK IDX RANDOM	DISK IDX DYNAMIC	WORK STATION	DISKETTE	FORMATFILE
PROCEDURE DIVISION																
OPEN	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
INPUT	R	O	O	O	O	O	O	O	O	.	O	O
OUTPUT	.	R	R	R	R	O	O	O	O	O	O	O	O	.	O	O
I-O	O	O	O	O	O	O	O	R	.	O
NO REWIND	I
REVERSED	I
EXTEND	O	O	O
CLOSE	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
REEL/UNIT	O
REMOVAL	O
NO REWIND	O
NO REWIND	O
WITH LOCK	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
READ	I	I	I	I	I	I	I	I	I	I	I	I
NEXT	I	.	.	.	I	.	.	.
FIRST	D	.	.	.
LAST	D	.	.	.
PRIOR	D	.	.	.
INTO	I	I	I	I	I	I	I	I	I	I	I	I
AT END	I	I	I	I	I	I	I	I	I	I	I	I
INVALID KEY	I	I	.	I	I	B	.	.	.
FORMAT	D	.	.	.	D	D	D	I	.	R
NEXT MODIFIED	B	.	.
SUBFILE	B	.	.
INDICATORS	I	.	.
TERMINAL	O	.	.
NO DATA	O	.	.
WRITE	.	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
FROM	.	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
INVALID KEY	O	O	O	O	O	O	B	.	.
ADVANCING	.	O	O	O	O
AT END-OF-PAGE	O
FORMAT	D	.	.	.	D	D	D	R	.	R
STARTING	O	.	.
ROLLING	O	.	.
INDICATORS	O	.	.
SUBFILE	B	.	.
TERMINAL	O	.	.

Figure I-1 (Part 2 of 3). File Structure Support

DEVICE TYPE	CARD READER	CARD PUNCH	CARD PRINT	CARD PUNCHPRINT	PRINTER	TAPE	DISK SEQ	DISK REL SEQ	DISK REL RANDOM	DISK REL DYNAMIC	DISK IDX SEQ	DISK IDX RANDOM	DISK IDX DYNAMIC	WORK STATION	DISKETTE	FORMATFILE
START	O	.	O	O	.	O	.	.	.
KEY	O	.	O	O	.	O	.	.	.
INVALID KEY	O	.	O	O	.	O	.	.	.
FORMAT	D	D	D	.	.	.
REWRITE	O	O	O	O	O	O	O	B	.	.
FROM	O	O	O	O	O	O	O	B	.	.
INVALID KEY	O	O	.	O	O	B	.	.
FORMAT	D	D	B	.	.
INDICATORS	B	.	.
SUBFILE	S	.	.
TERMINAL	O	.	.
DELETE	O	O	O	O	O	O	.	.	.
INVALID KEY	O	O	.	O	O	.	.	.
FORMAT	D	D	.	.	.
USE	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
EXCEPTION/ERROR	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
FOR DEBUGGING	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
COMMIT	D	D	D	D	D	D	D	.	.	.
ROLLBACK	D	D	D	D	D	D	D	.	.	.
ACQUIRE	O	.	.
DROP	O	.	.

Figure I-1 (Part 3 of 3). File Structure Support

Status Key 1 2	Meaning	When Set (CPF Exceptions Monitored, Condition Detected)
0	Successful Completion	
0	Successful Completion	No error condition occurred during the I-O operation.
1	At End of File	
0	End of file	CPF4740, CPF5070, CPF5001, CPF5025.
2	No modified subfile record found (IBM extension)	CPF5037.
2	Invalid Key	
1	Sequence error	REWRITE to an indexed file with sequential access and key for REWRITE \neq key from previous READ, or WRITE to an indexed file with sequential access and key values for succeeding writes are not in ascending sequence.
2	Duplicate key when duplicates are not allowed	CPF5008, CPF5009, CPF5010, CPF5026, CPF5034, CPF5084, CPF5085, or WRITE to an indexed file with sequential access and key values for succeeding writes are not in ascending sequence.
3	No record found	CPF5006, CPF5013, CPF5020.
4	Boundary violation	CPF5018, CPF5021, CPF5043, CPF5272, CPF5305 if organization is not sequential.
3	Permanent Error	
0	Permanent Error	CPF5101, CPF5102, CPF5113, CPF5120, CPF5129, CPF5143, CPF5156.
4	Boundary violation	CPF5116, CPF5018, CPF5272, CPF5305 if organization is sequential.

Figure I-2 (Part 1 of 5). Status Key Values and Meanings

Status Key 1 2	Meaning	When Set (CPF Exceptions Monitored, Condition Detected)
9 0	<p>Other Errors</p> <p>Other errors:</p> <ul style="list-style-type: none"> • File not found • Member not found • Level check error • Unexpected I-O exceptions 	<p>CPF4101 if a USE is applicable for the file.</p> <p>CPF4102 if a USE is applicable for the file.</p> <p>CPF4131.</p> <p>The following exceptions are monitored generically:</p> <p>CPF 4101 through CPF 4399 CPF 4501 through CPF 4699 CPF 4701 through CPF 4899 CPF 5001 through CPF 5099 CPF 5101 through CPF 5399</p> <p>These exceptions are caught and FILE STATUS is set to 90. If a USE procedure is applicable, it is executed. Otherwise, the program terminates and gives the operator the exception and the option to cancel, take a partial dump, or take a full dump.</p>

Figure I-2 (Part 2 of 5). Status Key Values and Meanings

Status Key 1 2	Meaning	When Set (CPF Exceptions Monitored, Condition Detected)
9	Other Errors (continued)	
1	Undefined or unauthorized access type	CPF2207, CPF4104, CPF5057, CPF5109, CPF5134, CPF5279.
2	Logic error: <ul style="list-style-type: none"> • File locked • File already open • I-O to closed file • READ after end of file • CLOSE on unopened file 	CPF4102, CPF4106, CPF5013, CPF5119, CPF5132, CPF5145, CPF5146, CPF5176, CPF5183.
4	No current record pointer	REWRITE/DELETE with sequential access, and last operation was not a successful READ.
5	Invalid or incomplete file information	(1) Duplicate keys specified in COBOL program, but indexed data base file created with unique key; or (2) Duplicate keys not specified in COBOL program, and indexed data base file created allowing duplicate keys.
A	Job has been cancelled in a controlled manner by CL command CNLJOB, TRMSBS, TRMCPF, or PWRDWSYS	CPF4741.
D	Record is locked	CPF5027, CPF5032.

Figure I-2 (Part 3 of 5). Status Key Values and Meanings

Status Key 1 2	Meaning	When Set (CPF Exceptions Monitored, Condition Detected)
9	Other Errors (continued)	
K	Invalid format-name; format not found	CPF5022, CPF5023, CPF5053, CPF5054, CPF5121, CPF5152, CPF5153, CPF5186, CPF5187,
M	Last record written to subfile	CPF5003.
N	Temporary (potentially recoverable) hardware I-O error	CPF4145, CPF4146, CPF4193, CPF4229, CPF4291, CPF4299, CPF4354, CPF4526, CPF4542, CPF4577, CPF4592, CPF4602, CPF4603, CPF4611, CPF4612, CPF4616, CPF4617, CPF4622, CPF4623, CPF4624, CPF4625, CPF4628, CPF4629, CPF4630, CPF4631, CPF4632, CPF5107, CPF5128, CPF5166, CPF5198, CPF5280, CPF5282, CPF5287, CPF5293, CPF5352, CPF5353, CPF5517, CPF5524, CPF5529, CPF5530, CPF5532, CPF5533.
P	OPEN failed because file cannot be placed under commitment control	CPF4285, CPF4293, CPF4326, CPF4327, CPF4328, CPF4329, CPF4330, CPF4331.

Figure I-2 (Part 5 of 5). Status Key Values and Meanings

Attribute Data Formats

The layouts and values of the attribute data are system dependent. The following formats are for the System/38.

Display Device Attribute Data

```
01  TERMINAL ATTRIBUTES.
    02  TERMINAL-TYPE PIC X.
    *   D - DISPLAY
    02  TERMINAL-SIZE PIC X.
    *   1 - 1920 CHARACTERS
    *   2 - 960 CHARACTERS
    02  TERMINAL LOCATION PIC X.
    *   L - LOCAL
    *   R - REMOTE
    02  TERMINAL-STATUS.
    *   03  FILLER PIC X.
    *       RESERVED
    03  ACQUIRE-STATUS PIC X.
    *   Y - ACQUIRED
    *   N - NOT ACQUIRED
    03  INVITE-STATUS PIC X.
    *   Y - INVITED
    *   N - NOT INVITED
    03  DATA-AVAILABLE-STATUS PIC X.
    *   Y - DATA IS AVAILABLE
    *       (ENTER OR COMMAND KEY PRESSED)
    *   N - DATA IS NOT AVAILABLE
    02  FILLER PIC X(9).
    *   RESERVED
```

Communications Device Attribute Data

```
01  COMMUNICATIONS-ATTRIBUTES.
    02  PROGRAM-DEVICE-STATUS PIC X.
    *   A - THE PROGRAM DEVICE IS NOT ACQUIRED
    *       (VALID FOR LU1, BSC, AND PEER)
    *   C - THE PROGRAM DEVICE IS ACQUIRED
    *       (VALID FOR LU1, BSC, AND PEER.
    *       FOR PEER, THIS IS THE SOURCE END OF THE SESSION.)
    *   R - THE PROGRAM DEVICE FOR THE TARGET END OF THE
    *       SESSION IS ACQUIRED (VALID ONLY FOR PEER)
    02  INVITE-STATUS PIC X.
    *   N - THIS PROGRAM DEVICE IS NOT INVITED
    *   I - THIS PROGRAM DEVICE IS INVITED BUT NO DATA
    *       HAS BEEN RECEIVED
    *   O - THIS PROGRAM DEVICE IS INVITED AND DATA IS READY
    02  SYNCHRONIZATION-LEVEL PIC X.
    *   C - SYNVL (*CONFIRM)
    *   N - SYNVL (*NONE)
    02  DEVICE-NAME PIC X(10).
    *   THE DEVICE NAME ASSOCIATED WITH THE PROGRAM DEVICE
```

OPEN-FEEDBACK and I-O-FEEDBACK Data Areas

OPEN-FEEDBACK

The OPEN-FEEDBACK area is part of the open data path (ODP) that contains information about the OPEN operation. This information is set during OPEN processing and is available as long as the file is open.

This area provides information about the file that the program is using. It contains:

- Information about the file that is currently open, such as:
 - File name
 - File type.
- Information that depends on the type of file that is opened, such as:
 - Printer size
 - Screen size
 - Diskette or tape labels.

I-O-FEEDBACK

The system updates the I-O-FEEDBACK area each time a block of records is transferred between CPF and the program. A block of records can contain one or more records.

The I-O-FEEDBACK area is not updated after each read or write for files in which multiple records are blocked and unblocked by COBOL. If the I-O-FEEDBACK area is needed after each read or write in the program, the user can do either of the following:

- Prevent the compiler from generating blocking and unblocking code by not satisfying one of the conditions listed under “Unblocking Input Records and Blocking Output Records” in Chapter 9.
- Specify SEQONLY(*NO) on the Override with Data Base File (OVRDBF) CL command.

Preventing the compiler from generating blocking and unblocking code is more efficient than specifying SEQONLY(*NO).

Even when the compiler generates blocking and unblocking code, certain CPF restrictions can cause blocking and unblocking to not be performed. In these cases, a performance improvement will not be realized. However, the I-O-FEEDBACK area will be updated after each read or write.

The I-O-FEEDBACK area contains information about the I-O operation. This area consists of a common area and a device-dependent area. The device-dependent area varies in length and content depending on the file type. This area follows the I-O-FEEDBACK common area and can be

obtained by specifying the receiving identifier large enough to include the common area and the appropriate device-dependent area.

The I-O-FEEDBACK area contains information about the last I-O operation, such as:

- Device name
- Device type
- AID character
- Error information for some devices.

See the *CPF Programmer's Guide* for a layout and description of the data areas contained in the OPEN-FEEDBACK and I-O-FEEDBACK areas.

Appendix J. Summary of Clauses and Statements

This appendix contains the following information:

- A general outline of a COBOL program by division, including the PROCESS statement
- A detailed format summary for each division in a COBOL program
- The symbols allowed in the PICTURE clause
- The assignment-names in the ASSIGN clause.

Conventions Used for Describing Statement Formats

When statement formats are described in this appendix, capitalized words, underlined words, braces, square brackets, and ellipsis have the following meanings:

- Reserved words are printed entirely in capital letters:
 - Required reserved words (keywords) are capitalized and underlined.
 - Optional reserved words are capitalized but not underlined.
- User-defined words are shown in lowercase letters.
- Braces { } enclosing listed items indicate (1) that exactly one of the enclosed stacked items must be specified and/or (2) when followed by an ellipsis, that the enclosed unit or item must be specified at least once.
- Square brackets [] indicate the enclosed item or unit can be used or omitted as required for the program. When two or more items are stacked within brackets, one or none of them can be specified. When followed by an ellipsis, the item or unit can be repeated.
- An ellipsis (...) indicates that the immediately preceding unit can occur one or more times in succession.

IBM extensions to American National Standard COBOL, X3.23-1974 are boxed like this sentence.

COBOL clauses and statements that are syntax-checked, but are treated as documentation by the System/38 COBOL compiler are boxed like this sentence.

COBOL Program Structure

Process Statement

PROCESS option-1 [option-2] . . . [option-n] [.]

Note: The PROCESS statement specifies compile-time options. It is not a COBOL source statement. If you want any options that are not default options of the PROCESS statement, this statement must precede the Identification Division header. PROCESS statement options and defaults are defined later in this appendix.

Identification Division

IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

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

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

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

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

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

Environment Division

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. source-computer-entry

OBJECT-COMPUTER. object-computer-entry

[SPECIAL-NAMES. special-names-entry]

[INPUT-OUTPUT SECTION.

FILE-CONTROL. {file-control-entry} . . .

[I-O-CONTROL. input-output-control-entry]

Data Division

DATA DIVISION.

[FILE SECTION.

[file-description-entry, { record-description-entry } . . .]

[sort-merge-file-description-entry, { record-description-entry } . . .]]

[WORKING-STORAGE SECTION.

[data-description-entry] . . .

[record-description-entry] . . .]

[LINKAGE SECTION.

[data-description-entry] . . .

[record-description-entry] . . .]

Procedure Division – Format 1 – Declaratives Section

PROCEDURE DIVISION [USING data-name-1 [, data-name-2] . . .] .

[DECLARATIVES.

{ section-name SECTION [segment-number] . use-sentence

[paragraph-name. [sentence] . . .] . . . } . . .

END DECLARATIVES.]

{ section-name SECTION [segment-number] .

[paragraph-name. [sentence] . . .] . . . } . . .

Procedure Division – Format 2

PROCEDURE DIVISION [USING data-name-1 [, data-name-2] . . .] .

{ paragraph-name. [sentence] . . . } . . .

DETAILED FORMATS

Identification Division Format

IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

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

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

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

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

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

Environment Division Formats

Configuration Section

CONFIGURATION SECTION.

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

OBJECT-COMPUTER. computer-name

[, MEMORY SIZE integer { WORDS
CHARACTERS
MODULES }]

[, PROGRAM COLLATING SEQUENCE IS alphabet-name]

[, SEGMENT-LIMIT IS segment-number] .

[SPECIAL-NAMES. [function-name-1 IS mnemonic-name] . . .

[function-name-2

{ IS mnemonic-name , ON STATUS IS condition-name-1 [, OFF STATUS IS condition-name-2]
IS mnemonic-name , OFF STATUS IS condition-name-2 [, ON STATUS IS condition-name-1]
ON STATUS IS condition-name-1 [, OFF STATUS IS condition-name-2]
OFF STATUS IS condition-name-2 [, ON STATUS IS condition-name-1] }

[, alphabet-name IS { STANDARD-1
NATIVE
literal-1 [{ THROUGH } literal-2
THRU
ALSO literal-3 [, ALSO literal-4] . . .]
[literal-5 [{ THROUGH } literal-6
THRU
ALSO literal-7 [, ALSO literal-8] . . .]] . . . }

[, CURRENCY SIGN IS literal-9]

[, DECIMAL-POINT IS COMMA] .]

Input-Output Section

Note: The keyword FILE-CONTROL appears only once at the beginning of the paragraph before the first file-control entry. The key word I-O-CONTROL appears only once at the beginning of the paragraph before the input-output-control entry.

```
[INPUT-OUTPUT SECTION.  
FILE-CONTROL. {file-control-entry} . . .  
[I-O-CONTROL. input-output-control-entry ] ] .
```

FILE-CONTROL Paragraph – Sequential File Entries (READER, PUNCH, PUNCHPRINT, PRINT, PRINTER, TAPEFILE, DISKETTE, FORMATFILE, DISK, DATABASE)

```
SELECT [OPTIONAL] file-name  
ASSIGN TO assignment-name-1 [ , assignment-name-2 ] . . .  
[ [RESERVE integer-1 [ AREA  
[ AREAS ] ] ]  
[ ORGANIZATION IS SEQUENTIAL ]  
[ ACCESS MODE IS SEQUENTIAL ]  
[ FILE STATUS IS data-name-1 ] .
```

FILE-CONTROL Paragraph – Relative (Direct) File Entries (DISK, DATABASE)

SELECT file-name

ASSIGN TO assignment-name-1 [, assignment-name-2] ...

[RESERVE integer-1 [AREA AREAS]]

ORGANIZATION IS RELATIVE

[ACCESS MODE IS { SEQUENTIAL [, RELATIVE KEY IS data-name-3] }
{ RANDOM } , RELATIVE KEY IS data-name-3 }]

[FILE STATUS IS data-name-1] .

FILE-CONTROL Paragraph – Indexed File Entries (DISK, DATABASE)

SELECT file-name

ASSIGN TO assignment-name-1 [, assignment-name-2] ...

[RESERVE integer-1 [AREA AREAS]]

ORGANIZATION IS INDEXED

[ACCESS MODE IS { SEQUENTIAL }
{ RANDOM }
{ DYNAMIC }]

RECORD KEY IS { EXTERNALLY-DESCRIBED-KEY } [WITH DUPLICATES]
data-name-2

[FILE STATUS IS data-name-1] .

FILE-CONTROL Paragraph – Sort or Merge File Entries

SELECT file-name ASSIGN TO assignment-name-1 [, assignment-name-2]

FILE-CONTROL Paragraph – TRANSACTION File Entries (WORKSTATION)

```
SELECT file-name  
  
  ASSIGN TO assignment-name-1 [ , assignment-name-2 ] ...  
  
  ORGANIZATION IS TRANSACTION  
  
  [ ACCESS MODE IS { SEQUENTIAL  
    DYNAMIC, RELATIVE KEY IS data-name-3 } ]  
  
  [ FILE STATUS IS data-name-1, [ data-name-5 ]  
  
  [ CONTROL-AREA IS data-name-6 ] .
```

I-O-CONTROL Paragraph

```
[ I-O-CONTROL,  
  [ RERUN ON assignment-name  
    EVERY integer-1 RECORDS OF file-name-1 ] ... ]  
  
  [ SAME [ RECORD  
    SORT  
    SORT-MERGE ] AREA FOR file-name-2 { , file-name-3 } ... ] ... ]  
  
  [ MULTIPLE FILE TAPE CONTAINS  
    file-name-4 [ POSITION integer-2 ]  
    [ file-name-5 [ POSITION integer-3 ] ] ... ] ... ]
```

```
[ COMMITMENT CONTROL FOR  
  file-name-6  
  [ , file-name-7 ] . . . ]
```

Data Division Formats

File Section Formats

FD Entry – Files (FORMATFILE, DATABASE, DISK, READER, PUNCH, PUNCHPRINT, PRINT)

[FD file-name
[BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS
CHARACTERS }]
[RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]
LABEL { RECORD IS
RECORDS ARE } { STANDARD
OMITTED }
[VALUE OF user-name-1 IS { data-name-1
literal-1 }
[, user-name-2 IS { data-name-2
literal-2 }] . . .]
[DATA { RECORD IS
RECORDS ARE } data-name-3 [, data-name-4] . . .]
{ record-description-entry } . . .] . . .

FD Entry – Files (DISKETTE)

```
[ FD file-name
  [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS
    { CHARACTERS } ]
  [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]
  [ LABEL { RECORD IS
    { RECORDS ARE } } { STANDARD
    { OMITTED } } ]
  [ VALUE OF user-name-1 IS { data-name-1
    { literal-1 } } ]
  [ , user-name-2 IS { data-name-2
    { literal-2 } } ] . . . ]
  [ DATA { RECORD IS
    { RECORDS ARE } } data-name-3 [ , data-name-4 ] . . . ]
  [ CODE-SET IS alphabet-name ] .
  { record-description-entry } . . . ] . . .
```

FD Entry – Files (TAPEFILE)

```

[ FD file-name
  [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS
    CHARACTERS } ]
  [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]
  [ LABEL { RECORD IS } { STANDARD
    RECORDS ARE } { OMITTED } ]
  [ VALUE OF user-name-1 IS { data-name-1
    literal-1 }
    [ , user-name-2 IS { data-name-2 }
    literal-2 } ] . . . ]
  [ DATA { RECORD IS }
    RECORDS ARE } data-name-3 [ , data-name-4 ] . . . ]
  [ CODE-SET IS alphabet-name ] .
  { record-description-entry } . . . ] . . .

```

FD Entry – Files (PRINTER)

[FD file-name

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

[RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]

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

VALUE OF user-name-1 IS { data-name-1
literal-1 }

[, user-name-2 IS { data-name-2
literal-2 }] . . .]

[DATA { RECORD IS
RECORDS ARE } data-name-3 [, data-name-4] . . .]

[LINAGE IS { data-name-5
integer-5 } LINES [, WITH FOOTING AT { data-name-6
integer-6 }]

[, LINES AT TOP { data-name-7
integer-7 }] [, LINES AT BOTTOM { data-name-8
integer-8 }]]

{ record-description-entry } . . .] . . .

FD Entry – TRANSACTION File

```
[ FD file-name  
  [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]  
  [ LABEL { RECORD IS } { STANDARD }  
    { RECORDS ARE } { OMITTED } ]  
  [ DATA { RECORD IS }  
    { RECORDS ARE } data-name-3 [ , data-name-4 ] . . . ] .  
  { record-description-entry } . . . ]
```

SD Entry

```
[ SD file-name  
  [ RECORD CONTAINS [ integer-1 TO ] integer-2 CHARACTERS ]  
  [ DATA { RECORD IS }  
    { RECORDS ARE } data-name-1 [ , data-name-2 ] . . . ] .  
  { record-description-entry } . . . ]
```

Working-Storage Section

Data Description Entry – Format 1 – Item Description

level-number { data-name-1
 FILLER }

[REDEFINES data-name-2]

[{ PICTURE
 PIC } IS character-string]

[[USAGE IS] { DISPLAY
 COMPUTATIONAL
 COMP
 COMPUTATIONAL-3
 COMP-3
 COMPUTATIONAL-4
 COMP-4
 INDEX }]

[[SIGN IS] { LEADING
 TRAILING } [SEPARATE CHARACTER]]

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

[{ ASCENDING
 DESCENDING } KEY IS data-name-4 [, data-name-5] . . .] . . .

[INDEXED BY index-name-1 [, index-name-2] . . .]]

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

[{ JUSTIFIED
 JUST } RIGHT]

[BLANK WHEN ZERO]

[VALUE IS literal] .

Data Description Entry – Format 2 – Regroup or Rename Data Items

66 data-name-1 RENAMES data-name-2 [{ THROUGH / THRU } data-name-3] .

Data Description Entry – Format 3 – Condition-Name Description

88 condition-name { VALUE IS / VALUES ARE } literal-1 [{ THROUGH / THRU } literal-2]
[literal-3 [{ THROUGH / THRU } literal-4]]

Boolean Data Description Entry – Format 4 – Boolean Data

level-number { data-name-1 / FILLER }
[REDEFINES data-name-2]
[{ PICTURE / PIC } IS 1]
[[USAGE IS] DISPLAY]
[OCCURS { integer-1 TO integer-2 TIMES DEPENDING ON data-name-3 } / integer-2 TIMES]
[INDEXED BY index-name-1 [, index-name-2] . . .]]
[{ INDICATOR / INDICATORS / INDIC } integer-3]
[[{ SYNCHRONIZED / SYNC } [LEFT / RIGHT]]]
[[{ JUSTIFIED / JUST } RIGHT]]]
[VALUE IS Boolean-literal] .

Linkage Section

Any data description entry clause, except the VALUE clause, can describe items in the Linkage Section. The VALUE clause can be specified only for level-88 items. See “Working-Storage Section” earlier in this appendix for data description entry clause formats.

Procedure Division Formats

Procedure Division Header

PROCEDURE DIVISION [USING data-name-1 [, data-name-2] . . .] .

Procedure Division Statements

ACCEPT Statement – Format 1

ACCEPT identifier [FROM mnemonic-name]

ACCEPT Statement – Format 2

ACCEPT identifier FROM { DATE
DAY
TIME }

ACCEPT Statement – Format 3 – Feedback

ACCEPT identifier FROM mnemonic-name
[FOR file-name]

ACCEPT Statement – Format 4 – Local Data Area

ACCEPT identifier-1 FROM mnemonic-name
[FOR { identifier-2
literal }]

ACCEPT Statement – Format 5 – TRANSACTION Attributes

ACCEPT identifier-1 FROM mnemonic-name
FOR { identifier-2 }
 literal FOR file-name]

ACQUIRE Statement – TRANSACTION File

ACQUIRE { identifier }
 literal FOR file-name

ADD Statement – Format 1

ADD { identifier-1 } [, identifier-2] . . . TO identifier-m [ROUNDED]
 literal-1 [, literal-2]
 [, identifier-n [ROUNDED]] . . . [ON SIZE ERROR imperative-statement]

ADD Statement – Format 2 – Giving

ADD { identifier-1 } , { identifier-2 } [, identifier-3] . . .
 literal-1 literal-2 [, literal-3]
 GIVING identifier-m [ROUNDED] [, identifier-n [ROUNDED]] . . .
 [ON SIZE ERROR imperative-statement]

ADD Statement – Format 3 – Corresponding

ADD { CORRESPONDING } identifier-1 TO identifier-2 [ROUNDED]
 CORR
 [ON SIZE ERROR imperative-statement]

ALTER Statement

ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2
[, procedure-name-3 TO [PROCEED TO] procedure-name-4] . . .

CALL Statement

CALL { identifier-1 } [USING data-name-1 [, data-name-2] . . .]
[ON OVERFLOW imperative-statement]

CANCEL Statement

CANCEL { identifier-1 } [, identifier-2] . . .
[literal-1] [, literal-2] . . .

CLOSE Statement – Format 1

CLOSE file-name-1 [{ REEL } [WITH NO REWIND]
[UNIT] [FOR REMOVAL]]
[WITH { NO REWIND }
[LOCK]]]
[, file-name-2 [{ REEL } [WITH NO REWIND]
[UNIT] [FOR REMOVAL]]] . . .
[WITH { NO REWIND }
[LOCK]]]] . . .

CLOSE Statement – Format 2 – TRANSACTION File

CLOSE file-name-1 [WITH LOCK]
[file-name-2 [WITH LOCK]] . . .

COMMIT Statement

COMMIT

COMPUTE Statement

COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] . . .
= arithmetic-expression [ON SIZE ERROR imperative-statement]

Declarative Procedures

[DECLARATIVES.
{ section-name SECTION [segment-number] . use-sentence
[paragraph-name. [sentence] . . . } . . . } . . .
END DECLARATIVES.]

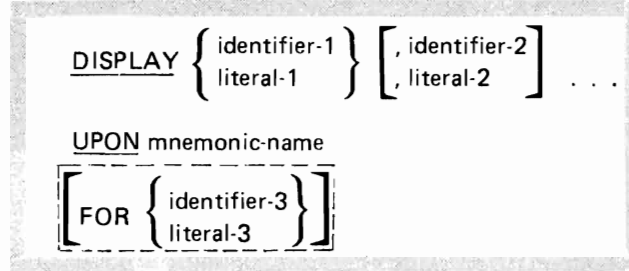
DELETE Statement

DELETE file-name RECORD [FORMAT IS { identifier }
literal]
[INVALID KEY imperative-statement]

DISPLAY Statement – Format 1

DISPLAY { identifier-1 } [, identifier-2] . . . [UPON mnemonic-name]
literal-1 } [, literal-2]

DISPLAY Statement – Format 2 – Local Data Area



The diagram shows the syntax for the DISPLAY statement in Format 2. It consists of three parts: the keyword `DISPLAY` followed by a list of identifiers and literals in curly braces, the keyword `UPON` followed by a mnemonic name, and an optional `FOR` clause in square brackets. The `FOR` clause contains an identifier and a literal in curly braces. Ellipses indicate that the list of identifiers and literals can contain more than two items.

```
DISPLAY { identifier-1 } [ , identifier-2 ] . . .  
UPON mnemonic-name  
[ FOR { identifier-3 }  
  literal-3 ]
```

DIVIDE Statement – Format 1

```
DIVIDE { identifier-1 } INTO identifier-2 [ ROUNDED ]  
[ , identifier-3 [ ROUNDED ] ] . . . [ ON SIZE ERROR imperative-statement ]
```

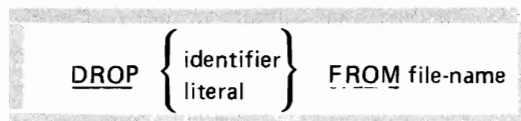
DIVIDE Statement – Format 2 – Giving

```
DIVIDE { identifier-1 } { INTO } { identifier-2 } GIVING identifier-3 [ ROUNDED ]  
[ , identifier-4 [ ROUNDED ] ] . . . [ ON SIZE ERROR imperative-statement ]
```

DIVIDE Statement – Format 3 – Giving, Remainder

```
DIVIDE { identifier-1 } { INTO } { identifier-2 } GIVING identifier-3 [ ROUNDED ]  
REMAINDER identifier-4 [ ON SIZE ERROR imperative-statement ]
```

DROP Statement



The diagram shows the syntax for the DROP statement. It consists of the keyword `DROP` followed by an identifier and a literal in curly braces, and the keyword `FROM` followed by a file name.

```
DROP { identifier }  
  literal FROM file-name
```

ENTER Statement

ENTER language-name [routine-name] .

EXIT Statement

paragraph-name. EXIT [PROGRAM] .

Note: The EXIT statement must be preceded by a paragraph-name, and it must be the only statement in the paragraph.

GO TO Statement – Format 1

GO TO [procedure-name-1]

GO TO Statement – Format 2 – Depending On (Conditional)

GO TO procedure-name-1 [, procedure-name-2] . . . , procedure-name-n
DEPENDING ON identifier

IF Statement

IF condition [THEN] { statement-1
NEXT SENTENCE } [{ ELSE statement-2
ELSE NEXT SENTENCE }]

INSPECT Statement – Format 1 – Tallying

INSPECT identifier-1 TALLYING

{ , identifier-2 FOR { { ALL
LEADING } { identifier-3 }
CHARACTERS { literal-1 } } [{ BEFORE
AFTER } INITIAL { identifier-4 }
{ literal-2 }] } . . . } . . .

INSPECT Statement – Format 2 – Replacing

INSPECT identifier-1 REPLACING

$$\left\{ \begin{array}{l} \text{CHARACTERS BY } \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{INITIAL } \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\} \right] \\ \left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \\ \text{FIRST} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-5} \\ \text{literal-3} \end{array} \right\} \text{BY } \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{INITIAL } \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\} \right] \end{array} \right\} \dots \dots \end{array} \right\}$$

INSPECT Statement – Format 3 – Tallying and Replacing

INSPECT identifier-1 TALLYING

$$\left\{ \begin{array}{l} \text{identifier-2 FOR } \left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \\ \text{CHARACTERS} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{literal-1} \end{array} \right\} \\ \left[\left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{INITIAL } \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \right] \end{array} \right\} \dots \dots$$

REPLACING

$$\left\{ \begin{array}{l} \text{CHARACTERS BY } \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{INITIAL } \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\} \right] \\ \left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \\ \text{FIRST} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-5} \\ \text{literal-3} \end{array} \right\} \text{BY } \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \\ \left[\left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{INITIAL } \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\} \right] \end{array} \right\} \dots \dots$$

MERGE Statement

MERGE file-name-1 ON { ASCENDING
DESCENDING } KEY data-name-1 [, data-name-2] . . .
[ON { ASCENDING
DESCENDING } KEY data-name-3 [, data-name-4] . . .] . . .
[COLLATING SEQUENCE IS alphabet-name]
USING file-name-2, file-name-3 [, file-name-4] . . .
{ OUTPUT PROCEDURE IS section-name-1 [{ THROUGH
THRU } section-name-2] }
{ GIVING file-name-5 }

MOVE Statement – Format 1

MOVE { identifier-1
literal } TO identifier-2 [, identifier-3] . . .

MOVE Statement – Format 2 – Corresponding

MOVE { CORRESPONDING
CORR } identifier-1 TO identifier-2

MULTIPLY Statement – Format 1

MULTIPLY { identifier-1 } BY identifier-2 [ROUNDED]
[, identifier-3 [ROUNDED]] . . . [ON SIZE ERROR imperative-statement]

MULTIPLY Statement – Format 2 – Giving

MULTIPLY { identifier-1 } BY { identifier-2 } GIVING identifier-3 [ROUNDED]
[, identifier-4 [ROUNDED]] . . . [ON SIZE ERROR imperative-statement]

OPEN Statement – Sequential Files

OPEN { INPUT file-name-1 [REVERSED]
[WITH NO REWIND]
[, file-name-2 [REVERSED]
[WITH NO REWIND]] . . .
OUTPUT file-name-3 [WITH NO REWIND]
[, file-name-4 [WITH NO REWIND]] . . .
I-O file-name-5 [, file-name-6] . . .
EXTEND file-name-7 [, file-name-8] . . . } . . .

OPEN Statement – TRANSACTION File

OPEN I-O file-name-1 [file-name-2] . . .

OPEN Statement – Indexed and Relative Files

$$\underline{\text{OPEN}} \left\{ \left\{ \begin{array}{l} \underline{\text{INPUT}} \\ \underline{\text{OUTPUT}} \\ \underline{\text{I-O}} \end{array} \right\} \text{file-name-1} \left[\text{, file-name-2} \right] \dots \right\} \dots$$

PERFORM Statement – Format 1

$$\underline{\text{PERFORM}} \text{ procedure-name-1} \left[\left\{ \begin{array}{l} \underline{\text{THROUGH}} \\ \underline{\text{THRU}} \end{array} \right\} \text{procedure-name-2} \right]$$

PERFORM Statement – Format 2 – Multiple Times

$$\underline{\text{PERFORM}} \text{ procedure-name-1} \left[\left\{ \begin{array}{l} \underline{\text{THROUGH}} \\ \underline{\text{THRU}} \end{array} \right\} \text{procedure-name-2} \right] \left\{ \begin{array}{l} \text{identifier-1} \\ \text{integer-1} \end{array} \right\} \underline{\text{TIMES}}$$

PERFORM Statement – Format 3 – Until Condition Satisfied

$$\underline{\text{PERFORM}} \text{ procedure-name-1} \left[\left\{ \begin{array}{l} \underline{\text{THROUGH}} \\ \underline{\text{THRU}} \end{array} \right\} \text{procedure-name-2} \right] \underline{\text{UNTIL}} \text{condition-1}$$

PERFORM Statement – Format 4 – Varying Index or Identifier

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ]  
                             [ THRU ]  
VARYING { identifier-1 } FROM { identifier-2 }  
          { index-name-1 }      { index-name-2 }  
                               { literal-2 }  
BY { identifier-3 } UNTIL condition-1  
    { literal-3 }  
[ AFTER { identifier-4 } FROM { identifier-5 }  
  { index-name-4 }          { index-name-5 }  
                           { literal-5 }  
BY { identifier-6 } UNTIL condition-2  
    { literal-6 }  
[ AFTER { identifier-7 } FROM { identifier-8 }  
  { index-name-7 }          { index-name-8 }  
                           { literal-8 }  
BY { identifier-9 } UNTIL condition-3 ] ]
```

READ Statement – Format 1 – Sequential Retrieval using SEQUENTIAL Access

```
READ file-name RECORD
```

```
[ INTO identifier-1 ]
```

```
[ FORMAT IS { identifier-2 } ]  
             { literal-1 } ]
```

```
[ AT END imperative-statement ]
```

READ Statement – Format 2 – Sequential Retrieval using DYNAMIC Access

READ file-name {
FIRST
LAST
NEXT
PRIOR
} RECORD

[INTO identifier-1]

[FORMAT IS { identifier-2 }
literal-1]

[AT END imperative-statement]

READ Statement – Format 3 – Random Retrieval

READ file-name RECORD [INTO identifier-1]

[FORMAT IS { identifier-2 }
literal-1]

[INVALID KEY imperative-statement]

READ Statement – Format 4 – TRANSACTION File (Nonsubfile)

```
READ file-name RECORD  
  
  [ INTO identifier-1 ]  
  
  [ FORMAT IS { identifier-2 }  
    { literal-1 } ]  
  
  [ TERMINAL IS { identifier-3 }  
    { literal-2 } ]  
  
  [ { INDICATOR  
    INDICATORS [ IS ]  
    INDIC [ ARE ] } identifier-4 ]  
  
  [ NO DATA imperative-statement-1 ]  
  
  [ AT END imperative-statement-2 ]
```

READ Statement – Format 5 – TRANSACTION File (Subfile)

```
READ SUBFILE file-name  
  
  [ NEXT MODIFIED ] RECORD  
  
  [ INTO identifier-1 ]  
  
  [ FORMAT IS { identifier-2 }  
    { literal-1 } ]  
  
  [ TERMINAL IS { identifier-3 }  
    { literal-2 } ]  
  
  [ { INDICATOR  
    INDICATORS [ IS ]  
    INDIC [ ARE ] } identifier-4 ]  
  
  [ INVALID KEY imperative-statement-1 ]  
  
  [ AT END imperative-statement-2 ]
```

RELEASE Statement

RELEASE record-name [FROM identifier]

RETURN Statement

RETURN file-name RECORD [INTO identifier] AT END imperative-statement

REWRITE Statement – Format 1

REWRITE record-name [FROM identifier-1]

[FORMAT IS { identifier-2 }
literal-1]

[INVALID KEY imperative-statement]

REWRITE Statement – Format 2 – TRANSACTION File (Subfile)

REWRITE SUBFILE record-name [FROM identifier-1]

[FORMAT IS { identifier-2 }
literal-1]
[TERMINAL IS { identifier-3 }
literal-2]
[{ INDICATOR [IS] identifier-4
INDICATORS [ARE]
INDIC }]
[INVALID KEY imperative-statement]

ROLLBACK Statement

[ROLLBACK]

SEARCH Statement – Format 1 – Selective Table Search

SEARCH identifier-1 [VARYING { identifier-2
index-name-1 }] [AT END imperative-statement-1]
WHEN condition-1 { imperative-statement-2
NEXT SENTENCE }
[WHEN condition-2 { imperative-statement-3
NEXT SENTENCE }] . . .

SEARCH Statement – Format 2 – Key Table Search

SEARCH ALL identifier-1 [AT END imperative-statement-1]
WHEN { data-name-1 { IS EQUAL TO
IS = } { identifier-3
literal-1
arithmetic-expression-1 } }
condition-name-1
[AND { data-name-2 { IS EQUAL TO
IS = } { identifier-4
literal-2
arithmetic-expression-2 } }] . . .
{ imperative-statement-2
NEXT SENTENCE }

SET Statement – Format 1

SET mnemonic-name-1 [, mnemonic-name-2] . . . TO { ON
OFF }

SET Statement – Format 2

SET condition-name-1 [, condition-name-2] . . . TO TRUE

SET Statement – Format 3

SET { identifier-1 [, identifier-2] . . . } TO { identifier-3
index-name-3
integer-1 }

SET Statement – Format 4

SET index-name-4 [, index-name-5] . . . { UP BY
DOWN BY } { identifier-4
integer-2 }

SORT Statement

SORT file-name-1 ON { ASCENDING
DESCENDING } KEY data-name-1 [, data-name-2] . . .
[ON { ASCENDING
DESCENDING } KEY data-name-3 [, data-name-4] . . .] . . .
[COLLATING SEQUENCE IS alphabet-name]
{ INPUT PROCEDURE IS section-name-1 [{ THROUGH
THRU } section-name-2] }
{ USING file-name-2 [, file-name-3] . . . }
{ OUTPUT PROCEDURE IS section-name-3 [{ THROUGH
THRU } section-name-4] }
{ GIVING file-name-4 }

START Statement

START file-name [KEY IS { EQUAL TO
=
GREATER THAN
>
NOT LESS THAN
NOT < } { EXTERNALLY-DESCRIBED-KEY
data-name-1 [, data-name-2] . . . }]
[FORMAT IS { identifier-1
literal-1 }]
[INVALID KEY imperative-statement]

STOP Statement

STOP { RUN }
 { literal }

STRING Statement

STRING { identifier-1 } [, identifier-2] . . . DELIMITED BY { identifier-3 }
 { literal-1 } [, literal-2] . . . { literal-3 }
 { SIZE }
 [, { identifier-4 } [, identifier-5] . . . DELIMITED BY { identifier-6 }
 { literal-4 } [, literal-5] . . . { literal-6 }
 { SIZE }] . . .
INTO identifier-7 [WITH POINTER identifier-8]
 [ON OVERFLOW imperative-statement]

SUBTRACT Statement – Format 1

SUBTRACT { identifier-1 } [, identifier-2] . . . FROM identifier-3 [ROUNDED]
 { literal-1 } [, literal-2] . . . [, identifier-4 [ROUNDED]] . . . [ON SIZE ERROR imperative-statement]

SUBTRACT Statement – Format 2 – Giving

SUBTRACT { identifier-1 } [, identifier-2] . . . FROM { identifier-3 }
 { literal-1 } [, literal-2] . . . { literal-3 }
GIVING identifier-4 [ROUNDED] [, identifier-5 [ROUNDED]] . . .
 [ON SIZE ERROR imperative-statement]

SUBTRACT Statement – Format 3 – Corresponding

SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2 [ROUNDED]
[ON SIZE ERROR imperative-statement]

UNSTRING Statement

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]
[, identifier-7 [, DELIMITER IN identifier-8] [, COUNT IN identifier-9]] . . .]
[WITH POINTER identifier-10] [TALLYING IN identifier-11]
[ON OVERFLOW imperative-statement]

USE Statement – EXCEPTION/ERROR Procedure – Format 1

USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1 [, file-name-2] . . . }
[ERROR] [INPUT] [OUTPUT] [I-O] [EXTEND]

USE Statement – EXCEPTION/ERROR Procedure (TRANSACTION) – Format 2

USE AFTER STANDARD { ERROR }
[EXCEPTION]
PROCEDURE ON { file-name-1 [file-name-2] . . . }
[I-O]

USE Statement – FOR DEBUGGING

USE FOR DEBUGGING ON { [ALL REFERENCES OF] identifier-1
file-name-1
procedure-name-1
ALL PROCEDURES }
[[ALL REFERENCES OF] identifier-2
file-name-2
procedure-name-2
ALL PROCEDURES] ...

WRITE Statement – Format 1 – Sequential Files

WRITE record-name [FROM identifier-1]
[{ BEFORE } ADVANCING { { identifier-2 } { LINE }
{ AFTER } { integer } { LINES } }
{ mnemonic-name }
{ PAGE }]
[AT { END-OF-PAGE } imperative-statement]
{ EOP }

WRITE Statement – Format 2 – Indexed, and Relative Files

WRITE record-name [FROM identifier-1]
[FORMAT IS { identifier-2 }
{ literal-1 }]
[INVALID KEY imperative-statement]

WRITE Statement – Format 3 – FORMATFILE Files

WRITE record-name [FROM identifier-1]

[FORMAT IS { identifier-2 }
 literal-1]

[{ INDICATOR
INDICATORS [IS] } identifier-3
 INDIC [ARE] }]

[AT { END-OF-PAGE
 EOP } imperative-statement]

WRITE Statement – Format 4 – TRANSACTION File (Nonsubfile)

WRITE record-name [FROM identifier-1]

FORMAT IS { identifier-2 }
 literal-1]

[TERMINAL IS { identifier-3 }
 literal-2]]

[STARTING AT LINE { identifier-4 }
 literal-3]]

{ BEFORE } ROLLING [LINES] { identifier-5 }
 { AFTER } [LINE] { literal-4 }]

[THROUGH] { identifier-6 } { UP
 THRU] { literal-5 } { DOWN }]

{ identifier-7 } [LINES
 literal-6] [LINE]]

[{ INDICATOR
INDICATORS [IS] } identifier-8
 INDIC [ARE] }]

WRITE Statement – Format 5 – TRANSACTION File (Subfile)

```
WRITE SUBFILE record-name [FROM identifier-1]  
  
FORMAT IS { identifier-2  
                  literal-1 }  
  
[TERMINAL IS { identifier-3  
                  literal-2 }]  
  
[ { INDICATOR  
   INDICATORS [IS  
   INDIC      ARE] } identifier-8 ]  
  
[INVALID KEY imperative-statement ]
```

Conditional Expressions

Class Condition

identifier IS [NOT] { NUMERIC
ALPHABETIC }

Condition-Name Condition

condition-name

Relation Condition

operand-1 IS [NOT] { GREATER THAN
LESS THAN
EQUAL TO
>
<
= } operand-2

Sign Condition

operand IS [NOT] { POSITIVE
NEGATIVE
ZERO }

Switch-Status Condition

condition-name

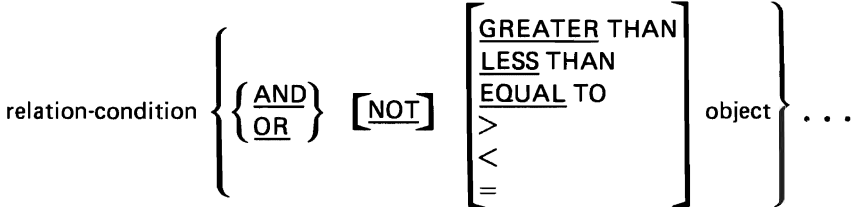
Negated Simple Condition

NOT simple-condition

Combined Condition

condition { { AND } condition } . . .

Abbreviated Combined Relation Condition



Qualification of Data Reference Formats

Data Item Reference

$$\left\{ \begin{array}{l} \text{data-name-1} \\ \text{condition-name} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right\} \text{data-name-2} \right] . . .$$

Procedure Name Reference

$$\text{paragraph-name} \left[\left\{ \begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right\} \text{section-name} \right]$$

COPY Library Reference

$$\text{text-name} \left[\left\{ \begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right\} \text{library-name} \right]$$

Note: library-name has the following format:
System/38 file name [-System/38 library name]

Subscripting

$$\left\{ \begin{array}{l} \text{data-name-1} \\ \text{condition-name} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right\} \text{data-name-2} \right] . . . \left(\text{subscript-1} \left[\text{,subscript-2} \left[\text{,subscript-3} \right] \right] \right)$$

Indexing

$$\left\{ \begin{array}{l} \text{data-name-1} \\ \text{condition-name} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right\} \text{data-name-2} \right] \dots \left(\left\{ \begin{array}{l} \text{index-name-1} \left[\left\{ \pm \right\} \text{literal-2} \right] \\ \text{literal-1} \end{array} \right\} \right. \\ \left. \left[\left\{ \begin{array}{l} \text{index-name-2} \left[\left\{ \pm \right\} \text{literal-4} \right] \\ \text{literal-3} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{index-name-3} \left[\left\{ \pm \right\} \text{literal-6} \right] \\ \text{literal-5} \end{array} \right\} \right] \right] \right) \right)$$

All Divisions

Copy Statement – Format 1

$$\text{COPY text-name } \left[\begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right] \text{System/38 file name } [-\text{System/38 library name}]$$

$$\left[\text{REPLACING } \left\{ \begin{array}{c} \text{==pseudo-text-1==} \\ \text{identifier-1} \\ \text{literal-1} \\ \text{word-1} \end{array} \right\} \text{BY } \left\{ \begin{array}{c} \text{==pseudo-text-2==} \\ \text{identifier-2} \\ \text{literal-2} \\ \text{word-2} \end{array} \right\} \right] \dots \left. \right]$$

COPY Statement – Format 2 – DDS Translate

$$\text{COPY } \left\{ \begin{array}{c} \text{DD-format-name} \\ \text{DD-ALL-FORMATS} \\ \text{DDS-format-name} \\ \text{DDS-ALL-FORMATS} \end{array} \right\} \left[\begin{array}{c} \text{-I} \\ \text{-O} \\ \text{-I-O} \end{array} \right] \left[\begin{array}{c} \text{-INDICATOR} \\ \text{-INDICATORS} \\ \text{-INDIC} \end{array} \right]$$

$$\left[\begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right] \text{System/38 file name } [-\text{System/38 library name}]$$

$$\left[\text{REPLACING } \left\{ \begin{array}{c} \text{==pseudo-text-1==} \\ \text{identifier-1} \\ \text{literal-1} \\ \text{word-1} \end{array} \right\} \text{BY } \left\{ \begin{array}{c} \text{==pseudo-text-2==} \\ \text{identifier-2} \\ \text{literal-2} \\ \text{word-2} \end{array} \right\} \right] \dots \left. \right]$$

PROCESS Statement

The PROCESS statement allows you to specify compile-time options unique to COBOL. The PROCESS statement must immediately precede the IDENTIFICATION DIVISION header. The format of the PROCESS statement is as follows:

```
PROCESS option-1 [option-2] . . . [option-n] [.]
```

The following list identifies PROCESS statement options. The options can appear in any order. Defaults are underlined.

<u>SOURCE</u> or SRC	<u>OPTIONS</u>
NOSOURCE or NOSRC	<u>NOOPTIONS</u>
XREF	<u>QUOTE</u>
<u>NOXREF</u>	<u>APOST</u>
<u>GEN</u>	<u>LIST</u>
NOGEN	<u>NOLIST</u>
<u>SEQUENCE</u>	<u>GENLVL(n) [n = 0 through 29]</u>
NOSEQUENCE	<u>GENLVL(29)</u>
VBSUM	FIPS(<u>NO</u> ,L,LI,HI,H)
<u>NOVBSUM</u>	
LINENUMBER	<u>FLAG(00)</u>
NUMBER	FLAG(n) [n = 0 through 99]
<u>NONUMBER</u>	
MAP	
<u>NOMAP</u>	

Symbols Allowed in the PICTURE Clause

Symbol	Meaning
A	Alphabetic character or space
B	Space insertion character
P	Decimal scaling position (not counted in size of data item)
S	Operational sign (not counted in size of data item unless a SIGN clause with optional SEPARATE CHARACTER phrase is specified)
V	Assumed decimal point (not counted in size of data item)
X	Alphanumeric character (any from the EBCDIC set)
Z	Zero suppression character
9	Numeric character
1	Boolean character
0	Zero insertion character
/	Slash insertion character
,	Comma insertion character
.	Decimal point or period editing control character
+	Plus sign insertion editing control character
-	Minus sign editing control character
CR	Credit editing control character
DB	Debit editing control character
*	Check protect insertion character
CS	Currency sign insertion character (default is \$).

Assignment-Names in the ASSIGN Clause

device-system/38 file name [-attribute]

device:	READER PUNCH PUNCHPRINT PRINT PRINTER TAPEFILE DISKETTE DISK DATABASE WORKSTATION FORMATFILE
system/38 file name:	1-10 character system-name
attribute:	- hopper [-association] SI (for separate indicator area)
hopper:	P or S (for card devices only)
association:	0-9 (for card devices only)



Glossary

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 files are referenced by the computer. Access can be sequential (records are referred to one after another in the order in which they appear on the file), it can be random (the individual records can be referred to in a nonsequential manner), or it can be dynamic (records can be accessed sequentially or randomly, depending on the form of the specific input/output request).

access path. The means by which CPF provides a logical organization to the data in a data base file so that the data can be processed by a program. See also *arrival sequence access path* and *keyed sequence access path*.

actual decimal point. The physical representation – using either of the decimal point characters (. or ,) – of the decimal point position in a data item. The actual decimal point appears in printed reports and requires a storage position in a data item. Contrast with *assumed decimal point*.

alias name. A DDS keyword, which allows a DDS field to be referenced by an alternate name (an alias).

alphabet-name. A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, which names a character set and/or collating sequence.

alphabetic character. A character that is one of the 26 uppercase characters of the alphabet, or a space.

alphanumeric character. Any character in the computer's character set.

alphanumeric edited character. An alphanumeric data item whose PICTURE character string contains at least one B, 0, or /.

American National Standard Code for Information Interchange. The standard code used for information interchange between data processing systems, data communications systems, and associated equipment. The code uses a coded character set consisting of 7-bit coded characters (8 bits including parity check). The set consists of control characters and graphic characters. Abbreviated ASCII.

American National Standards Institute. An organization sponsored by the Computer and Business Equipment Manufacturers Association for the purpose of establishing voluntary industry standards. Abbreviated ANSI.

ANSI. See *American National Standards Institute*.

arithmetic expression. A statement containing any combination of data-names, numeric literals, and figurative constants, joined together by one or more arithmetic operators in such a way that the statement as a whole can be reduced to a single numeric value.

arithmetic operator. A symbol that indicates the arithmetic operation to be performed. The arithmetic operators include: + (addition), - (subtraction), * (multiplication), / (division), and ** (exponentiation).

arrival sequence access path. An access path that is based on the order in which records are stored in a physical file. Contrast with *keyed sequence access path*.

ascending key. The values by which data is ordered from the lowest value to the highest value of the key in accordance with the rules for comparing data items. Contrast with *descending key*.

ascending key sequence. The arrangement of data in an order from the lowest value of the key field to the highest value of the key field. Contrast with *descending key sequence*.

ASCII. See *American National Standard Code for Information Interchange*.

assignment-name. A word that associates a file-name with an external device.

assumed decimal point. A logical decimal point position that does not occupy a storage position in a data item. It is used by the compiler to align a value properly for calculation. Contrast with *actual decimal point*.

AT END condition. A condition that occurs at the following times: during the execution of a READ statement for a sequentially accessed file; during the execution of a RETURN statement when no next logical record exists for the associated sort or merge file; during the execution of a SEARCH statement when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

auxiliary storage. All addressable storage other than main storage. Auxiliary storage is located in the system's nonremovable disk enclosures.

binary item. A numeric data item that is represented internally in binary notation (that is, as a number in the base 2); internally, each bit of the item is a binary digit with the sign as the leftmost bit.

binary synchronous communications. A form of communications line control that uses transmission control characters to control the transfer of data over a communications line. Abbreviated BSC. Contrast with *synchronous data link control*.

block. A unit of data that is moved into or out of the computer.

Boolean data type. A category of data items that are limited to a value of 1 or 0.

Boolean literal. A literal composed of a Boolean character enclosed in double quotes and preceded by a B; for example, B"1".

bottom margin. A blank area that follows the page body.

boundary violation. An attempt to write beyond the externally defined boundaries of a sequential file.

breakpoint. A place in a program (specified by a command or a condition) where the system halts execution and gives control to the work station user or to a specified program.

BSC. See *binary synchronous communications*.

BSC file. A device file created by the user to support BSC. Contrast with *communications file*.

buffer. A portion of main storage into which data is read or from which it is written.

called program. A program whose execution is requested by another program (a calling program) or by a command.

calling program. A program that requests the execution of another program (a called program).

character. Any letter, digit, or other symbol in the data character set that is part of the organization, control, or representation of data.

character literal. A symbol, quantity, or constant in a source program that is itself data, instead of a reference to data. Contrast with *numeric literal*.

character set. All the valid characters for a programming language or for a computer system.

character string. A sequence of characters that form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

class condition. A logical condition that states that the content of an item is all alphabetic or all numeric.

clause. An ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry. There are three types of clauses: data, environment, and file.

collating sequence. The order each character holds in relation to other characters according to the bit structure.

column. A character position measured horizontally within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

combined condition. A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

comment. A word or statement in a program, command, or file that serves as documentation instead of as instructions. A comment is ignored by a compiler.

comment line. A source program line that is not translated by the compiler. The comment line can be used to document the program.

comment-entry. An entry in the Identification Division that is not translated by the compiler.

commitment boundary. In COBOL, a point established by the successful execution of each COMMIT or ROLLBACK statement. If no COMMIT or ROLLBACK has been issued, a commitment boundary is established by the first OPEN of any file under commitment control.

commitment control. An environment, defined in COBOL through the I-O-CONTROL Paragraph, that allows the synchronizing of data base changes through a COMMIT statement, or the canceling of data base changes through the ROLLBACK statement, and that provides different levels of locking for records being changed.

common key. The key fields that are common to all record formats in the file starting with the first key field (the most significant) and ending with the last key field (the least significant).

communications file. A device file created by the user to support LU1 SDLC communications. Contrast with *BSC file*.

communications device. A BSC, LU1, or PEER device used through a BSC, communications, or mixed file. In COBOL, these files are defined as ORGANIZATION IS TRANSACTION.

compilation. Translation of a source program (such as RPG or COBOL specifications) into an executable program.

compile time. The time during which a source program is translated by a compiler into an executable program.

compiler. A program that translates a source program into an executable program.

compiler-directing statement. A COBOL statement that causes the compiler to take a specific action during compilation, rather than causing the program to take a particular action during execution.

complex condition. A condition in which one or more logical operators (AND, OR, or NOT) act upon one or more conditions. Complex conditions include negated simple conditions, combined conditions, and negated combined conditions.

compound condition. A statement that tests two or more relational expressions. It may be true or false.

computer-name. A system-name that identifies the computer upon which the program is to be compiled or run.

condition. An expression in a program for which a truth value can be determined at execution time. Conditions include the simple conditions (relation condition, class condition, condition-name condition, switch-status condition, sign condition) and the complex conditions (negated simple conditions, combined conditions, negated combined conditions).

condition-name. A name assigned to a specific value, set of values, or range of values within the complete set of values that a conditional variable can have.

condition-name condition. A condition that states that the value of a conditional variable is a member of the set of values assigned to a condition-name associated with the conditional variable.

conditional expression. A simple condition or a complex condition specified in an IF, PERFORM, or SEARCH statement. See also *simple condition* and *complex condition*.

conditional statement. A statement that causes the truth value of a condition to be determined and that controls program execution depending on this truth value.

conditional variable. A data item for which one or more values have a condition-name assigned to them.

CONFIGURATION SECTION. A section of the Environment Division of the COBOL program. It describes the overall specifications of computers.

connective. A word or a punctuation character that associates a data-name, paragraph-name, condition-name, or text-name with its qualifier; links two or more operands in a series; or forms a conditional expression.

CONSOLE. A COBOL function-name associated with the operator's keyboard/display.

constant. Data that has an unchanging, predefined value to be used in processing. A constant does not change during the execution of a program, but the contents of a field or variable can. See also *literal*.

contiguous items. Consecutive elementary or group items in the Data Division that are contained in a single data hierarchy.

control language. The set of all commands with which a user requests functions. Abbreviated CL.

currency sign. The character \$.

currency symbol. The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present, the currency sign is used. See *currency sign*.

current record. The record that is available in the record area associated with the file.

current record pointer. A conceptual entity that is used in sequential retrieval of the next record.

data base. The collection of all data base files stored in the system.

data clause. A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

data communications file. A generic term for a communications file or a BSC file. See also *communications file* and *BSC file*.

data description entry. A Data Division entry that describes the characteristics of a data item.

data description specifications. A description of the user's data base or device files that is entered into the system using a fixed-form syntax. The description is then used to create files. Abbreviated DDS.

DATA DIVISION. One of the four main parts of a COBOL program. The Data Division describes the files to be used in the program and the records contained within the files. It also describes any internal Working-Storage records that will be needed.

data item. A character or a set of contiguous characters (excluding literals) defined as a unit of data by the COBOL program.

data-name. A user-defined word that names a data item. When used in the general formats, data-name represents a word that can be neither subscripted, indexed, nor qualified unless specifically permitted by the rules of that format. An index-name is not a data-name. See also *identifier*.

DD. See *Alias Name*.

DDS. See *data description specifications*.

debugging line. A COBOL statement executed only when the WITH DEBUGGING MODE clause is specified. Debugging lines can be used to help determine the cause of an error.

debugging section. A declaratives section that receives control when an identifier, file-name, or procedure-name is encountered in the Procedure Division.

declarative-sentence. A compiler-directing sentence that specifies when a debugging section or an exception/error procedure is to be executed.

Declaratives. A set of one or more special-purpose sections, written at the beginning of the Procedure Division, and which can be used for input/output error checking or debugging.

delimiter. A character or a sequence of contiguous characters that identifies the end of a string of characters. A delimiter separates a string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

descending key. The values by which data is ordered from the highest value to the lowest value of the key, in accordance with the rules for comparing data items. Contrast with *ascending key*.

descending key sequence. The arrangement of data in order from the highest value of the key field to the lowest value of the key field. Contrast with *ascending key sequence*.

device file. An object that contains a description of how input data is to be presented to a program from an external device and/or how output data is to be presented to the external device from the program. External devices can be work stations, card devices, printers, diskette magazine drives, magazine tape drives, or a communications link.

digit. Any of the numerals from 0 through 9.

direct file. See *relative file*.

Distributed Data Management (DDM). A program product that allows an application program or user on a source system to access data files on remote systems connected by a communications network that also uses DDM.

division. One of the four major parts in a COBOL program: Identification, Environment, Data, and Procedure.

division header. The COBOL reserved words and punctuation that indicate the beginning of one of the four divisions of a COBOL program.

dynamic access. An access mode in which records can be read from or written to a file in a nonsequential manner (see *random access*) and read from a file in a sequential manner (see *sequential access*) during the scope of the same OPEN statement.

EBCDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

editing character. A single character or a fixed 2-character combination used to format output.

elementary item. A data item that is described as not being logically subdivided.

entry. Any descriptive set of consecutive clauses terminated by a period and written in the Identification, Environment, or Procedure Division of a COBOL source program.

ENVIRONMENT DIVISION. One of the four main parts of a COBOL program. The Environment Division describes the computers upon which the source program is compiled and those on which the object program is executed; it also provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

exception. The occurrence of a machine or user-defined condition that can be monitored for and that is directly associated with the execution of a particular function within a program. Exceptions generally represent an abnormality detected by the machine or by a program.

executable program. The set of machine language instructions that is the output from the compilation of a source program. The actual processing of data is done by the executable program.

execution time. The time during which the instructions of a computer program are executed by a processing unit.

exponent. A number, indicating to which power another number (the base) is to be raised. In COBOL, exponentiation is indicated with the symbol ** followed by an exponent.

EXTEND mode. An open mode in which records are added to the end of a sequential file.

external decimal item. See *zoned decimal item*.

externally described data. Data contained in a file for which the fields in the records are described to CPF, by using data description specifications, when the file is created. The field descriptions can be used by the program when the file is processed. Contrast with *program-described data*.

externally described file. In S/38 COBOL, a file for which one or more COPY statements, DDS format, are coded as part of the record description entry (or entries) for the file. These COPY statements provide the program with the description of the file used by CPF to create and process the file. Contrast with *program described file*.

figurative constant. A reserved word that represents a numeric or character value, or a string of repeated values. The word can be used instead of a literal to represent the value.

FILE-CONTROL. The name and header of an Environment Division paragraph in which the data files for a given source program are named and assigned to specific input/output devices.

file description. An entry in the File Section of the Data Division that provides information about the identification and physical structure of a file.

file-name. A name, associated with a file, defined in a file description entry or in a sort-merge file description entry.

file organization. The permanent file structure established at the time that a file is created.

FILE SECTION. A section of the Data Division that contains descriptions of all externally stored data (or files) used in a program. Such information is given in one or more file description entries.

file separator. The pages or cards to be produced at the beginning of each output file to separate the file from the other files being spooled to an output device.

function-name. A name, defined by IBM, that identifies system logical units, system-supplied information, printer and card punch control characters, or program switches.

group item. A named set of contiguous elementary or group items.

hierarchy. A hierarchy is a set of entries that includes all subordinate entries to the next equal or higher level number.

IDENTIFICATION DIVISION. One of the four main parts of a COBOL program. The Identification Division identifies the source program and the object program and, in addition, may include such documentation as the author's name, the installation where written, the date written, and so on.

identifier. A data-name that is unique or is made unique by a combination of qualifiers, subscripts, and/or indexes.

imperative statement. A statement that specifies that an action is to be taken unconditionally. An imperative statement can consist of a series of imperative statements.

implementor-name. An IBM-defined name that includes assignment-names, function-names, and user-names.

index. A computer storage position or register, the contents of which identify a particular element in a set of elements.

index data item. A data item in which the contents of an index can be saved.

index-name. A user-defined word that names an index. An index-name is not a data-name.

indexed data-name. A data-name followed by one or more index-names, enclosed in parentheses, which is used to reference an element or a set of elements in a table.

indexed file. A data base file whose access path is built on key values. Each record in the file is identified by a key field.

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

indicator. An internal switch used by a program to remember when a certain event occurs and what to do when the event occurs.

input file. A file that is opened in the input mode.

input mode. An open mode in which records can be read from the file.

input-output file. A file that is opened in the I-O mode.

INPUT-OUTPUT SECTION. In the Environment Division, the section that names the files and external media needed by an object program. It also provides information required for the transmission

and handling of data during the execution of an object program.

INPUT PROCEDURE. A procedure that provides special processing of records when they are released to the sort function.

integer. A numeric data item or literal that does not include any character positions to the right of the decimal point. Where the term integer appears in formats, integer must be an unsigned numeric literal and must be nonzero unless the rules for that format explicitly state otherwise.

internal decimal item. See *packed decimal item*.

INVALID KEY condition. An execution-time condition in which the value of a key for an indexed or relative file does not give a correct reference to the file.

invited device. A work station or communications device that was written to using a DDS format that had the INVITE option specified. For multiple device files, the READ statement will read from any invited program device if no particular program device is specified for input using the TERMINAL phrase and no specific FORMAT is requested.

I-O-CONTROL. The name and the header for an Environment Division paragraph in which program requirements for specific input/output techniques are specified. These techniques include rerun checkpoints, the sharing of same areas by several data files, and the use of a storage-resident cylinder index.

I-O mode. An open mode where records can be read from, written to, or deleted from the file.

job separator. The pages or cards placed at the beginning of the output for each job that has spooled file entries on the output queue. Each separator contains information that identifies the job such as its name, the job user's name, the job number, and the time and date the job was executed.

key. A data item that identifies the location of a record, or a set of data items that is used to place data in ascending or descending sequence.

key field. A field in a record whose contents are used to sequence the records of a particular type within a file member.

keyword. A reserved word that is required by the syntax of a COBOL statement or entry.

keyed sequence access path. An access path to a data base file that is ordered on the contents of key

fields contained in the individual records. Contrast with *arrival sequence access path*.

language-name. A system-name that specifies a particular programming language.

level indicator. Two alphabetic characters, FD or SD, that identify the type of file description entry.

level-number. A numeric character (1 through 9) or a 2-character set (01 through 49, 66, 77, 88) that begins a data description entry, and establishes its level in a data hierarchy. Level-numbers 66, 77, and 88 identify special properties of a data description entry.

library. An object that serves as a directory to other objects. A library is used to group related objects and to find objects by name when they are used. The system-recognized identifier for the object type is *LIB.

library-name. A user-defined word that names a library.

LINKAGE SECTION. A section of the Data Division that describes data made available from another program.

literal. A character string whose value is given by the characters themselves. For example, the numeric literal 7 has the value 7, and the character literal 'CHARACTERS' has the value CHARACTERS. See also *character literal, constant, and numeric literal*.

local data area. An area automatically created by the system for each job. The local data area is defined outside the COBOL program as 1024 bytes of character data. The local data area initially has the same value as the submitting job's local data area or, if there was no submitting job, has an initial value of blanks.

logical file. A description of how data is to be presented to or received from a program. This type of data base file contains no data, but it provides an ordering and format for one or more physical files. Contrast with *physical file*.

logical operator. A COBOL reserved word that defines the logical connection between conditions or negates a condition: OR (logical connective – either or both), AND (logical connective – both), and NOT (logical negation).

logical record. The most inclusive data item. The level number for a logical record is 01.

main program. The highest level program involved in a run unit.

main storage. All storage in a computer from which instructions can be executed directly.

merge file. The temporary file that contains all the records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

mixed file. A CPF device file that supports: one or more work stations, one or more communications devices, or any combination of work stations and communications devices. A mixed file is processed in COBOL by a file with ORGANIZATION IS TRANSACTION.

mnemonic-name. A user-defined word associated with a function-name in the Environment Division.

mode. See *access mode*.

multiple device file. A device file that was created with the maximum number of program devices greater than one. Display files or mixed files can be multiple device files. Contrast with *single device file*.

name. A word that defines a COBOL operand. A name is composed of not more than 30 characters.

native character set. The default character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

native collating sequence. The default 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.

nest. To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another subroutine (the nesting subroutine).

next executable sentence. The sentence to which control is transferred after execution of the current statement is complete.

next executable statement. The statement to which control is transferred after execution of the current statement is complete.

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

noncontiguous item. A data item in the Working-Storage Section of the Data Division that bears no relationship with other data items.

nonnumeric item. A data item that is alphanumeric, alphabetic, or Boolean.

nonnumeric literal. A character string bounded by quotation marks, which literally means itself. See also *literal*.

numeric character. Any one of the digits 0 through 9.

numeric edited item. A numeric item whose PICTURE character-string contains valid editing characters.

numeric item. A data item that must be numeric. If signed, the item can also contain a representation of an operational sign.

numeric literal. The actual numeric value to be used in processing, instead of the name of a field containing the data. A numeric literal can contain any of the numeric digits 0 through 9, a sign (plus or minus), and a decimal point. Contrast with *character literal*.

OBJECT-COMPUTER. The name of an Environment Division paragraph in which the computer upon which the program will be run is described.

open mode. The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

operand. The object of a verb or an operator; that is, an operand is the data or equipment governed or directed by a verb or operator.

operational sign. An algebraic sign associated with a numeric data item or a numeric literal that indicates whether the item is positive or negative.

optional word. A reserved word included in a specific format only to improve the readability of a COBOL statement or entry.

output file. A file that is opened in either output mode or extend mode.

output mode. An open mode in which records can be written to a file.

OUTPUT PROCEDURE. A procedure that provides special processing of records when they are returned from the sort or merge function.

overflow condition. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

overlay. To use the same area of storage for more than one procedure.

packed decimal format. Representation of a decimal value in which each byte within a field represents two numeric digits except the rightmost byte, which contains one digit in bits 0 through 3 and the sign in bits 4 through 7. For all other bytes, bits 0 through 3 represent one digit; bits 4 through 7 represent one digit. For example, the decimal value +123 is represented as 0001 0010 0011 1111. Contrast with *zoned decimal format*.

packed decimal item. A numeric data item that is represented internally in packed decimal format.

paragraph. In the Procedure Division, a paragraph-name followed by a period and a space and by zero, one, or more sentences. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more sentences.

paragraph header. A reserved word, followed by a period and a space that indicates the beginning of a paragraph in the Identification and Environment Divisions.

paragraph-name. A user-defined word that identifies and begins a paragraph in the Procedure Division.

parameter. Data passed to or received from another program.

phrase. An ordered set of one or more consecutive COBOL character-strings that forms part of a clause or a Procedure Division statement.

physical file. A description of how data is to be presented to or received from a program and how data is actually stored in the data base. A physical file contains one record format and one or more members. Contrast with *logical file*.

physical record. A unit of data that is moved into or out of the computer. Same as *block*.

procedure. One or more successive paragraphs or sections within the Procedure Division, which direct the computer to perform some action or series of related actions.

PROCEDURE DIVISION. One of the four main component parts of a COBOL program. The Procedure Division contains instructions for solving a problem. The Procedure Division may contain imperative-statements, conditional statements, paragraphs, procedures, and sections.

procedure-name. A paragraph-name or a section-name in the Procedure Division.

process. Any operation or combination of operations on data.

program-described data. Data contained in a file for which the fields in the records are described in the program that processes the file. Contrast with *externally described data*.

program described file. In System/38 COBOL, a file that does *not* have any COPY statement, DDS format, coded as part of the record description entry for the file. The fields in the file's records are described only in the program that processes the file. Contrast with *externally described file*.

program device. A symbolic mechanism that a program uses instead of a real device (identified by the device name) to access the devices in a file. When the program uses a program device, the system redirects the operation to the appropriate real device. With the exception of mixed files, the name of the program device is the same as the name of the real device; for mixed files, the name of the program device may differ from the name of the real device.

program-name. A user-defined word that identifies a COBOL source program.

pseudo-text. A sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters. Pseudo-text is used in the COPY REPLACING statement for replacing text strings.

pseudo-text delimiter. Two contiguous equal signs (= =) used to delimit pseudo-text.

punctuation character. A character used to separate COBOL elements or to identify a particular type of COBOL element: a comma, semicolon, period, quotation mark, left or right parenthesis, or space.

qualified data-name. An identifier that is composed of a data-name followed by one or more

sets of either of the connectives OF or IN followed by a data-name qualifier.

qualifier. A name used to uniquely identify another name. Group data-names, section-names, and library-names can be used as qualifiers to form qualified names.

random access. An access mode in which specific records can be read from, written to, or deleted from a file in a nonsequential manner.

read-from-invited-program-devices operation. An input operation that waits for input from any one of the invited program devices for a user-specified time. Contrast with *read-from-one-program-device operation*.

read-from-one-program-device operation. An input operation that will not complete until the specified device has responded with input. Contrast with *read-from-invited-program-devices operation*.

record. A set of one or more related data items that are grouped for processing. Records can be defined for an input/output device or for internal processing. See also *logical record*.

record area. A storage area in which a record described in a record description entry in the File Section is processed.

record description entry. The total set of data description entries associated with a particular record.

record key. A key whose contents identify a record within an indexed file.

relation character. One of the characters that expresses a relationship between two operands: = (equal to), > (greater than), < (less than).

relation condition. A condition that relates two arithmetic expressions and/or data items.

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 to construct a relation condition.

relative file. A file with a relative organization. Same as *direct file*.

relative key. An unsigned integer data item that can be used directly by the system to locate a record in a file. Same as *relative record number*.

relative organization. The file structure in which each record is uniquely identified by a positive

integer value that specifies the record's ordinal position in the file.

relative record number. A number that specifies the location of a record in relation to the beginning of a data base file member or subfile. For example, the first record in a data base file member or subfile has a relative record number of 1.

reserved word. A predefined word used in a COBOL source program for syntactical purposes. It must not appear in a program as a user-defined name or system-name.

routine. A set of statements in a program that causes the computer to perform an operation or series of related operations.

run unit. A set of one or more programs that function as a unit at execution time to provide a problem solution. A run unit starts with the first COBOL program in the invocation stack and includes all programs (of any kind) that are below it in the invocation stack.

section. A set of zero, one, or more paragraphs or entries, called a section body, 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 period and a space, that indicates the beginning of a *section* in the Environment, Data, or Procedure Division.

section-name. A user-defined word that names a section in the Procedure Division.

sector. The addressable unit into which each track on a diskette is divided.

sentence. A sequence of one or more statements; the last statement ends with a period followed by a space.

separator. A punctuation character used to delimit character strings. See also *file separator* and *job separator*.

sequential access. An access mode in which records are read from, written to, or deleted from a file based on the logical order of the records in the file.

sequential processing. The processing of logical records in the order in which records are accessed.

serial search. A search in which the members of a set are consecutively examined, beginning with the first member and ending with the last member.

sign condition. A condition that states that the algebraic value of a data item is less than, equal to, or greater than zero.

simple condition. Any single condition chosen from the set: relation condition, class condition, condition-name condition, switch-status condition, and sign condition.

single device file. A device file created with only one program device defined for it. Printer files, card files, diskette files, tape files, communications files, and BSC files are single device files. Display files and mixed files created with a maximum number of one program device are also single device files. Contrast with *multiple device file*.

sort file. A temporary file that contains all the 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 that describes a sort file or a merge file.

SOURCE-COMPUTER. The name of an Environment Division paragraph describing the computer upon which the source program will be compiled.

source program. A set of instructions, written in a programming language such as RPG or COBOL, that represents a particular job as defined by a programmer. A source program is used as input to the compiler to create an executable program.

special character. A character that is neither numeric nor alphabetic. Special characters in COBOL include: + - * / = \$, . ") (; < >

special-character word. A reserved word that is an arithmetic operator or a relation character.

SPECIAL-NAMES. The names of an Environment Division paragraph and the paragraph itself in which names supplied by IBM are related to mnemonic-names specified by the programmer. In addition, this paragraph can be used to exchange the functions of the comma and the period or to specify a substitution character for the currency sign in the PICTURE string.

special registers. Compiler-generated data items used to store information produced by specific COBOL features (for example, the DEBUG-ITEM special register).

spooled file. A generic term for three types of files: a device file that provides access to an inline data file or that creates a spooled output file, an inline data file, or a spooled output file.

standard data format. The format in which data is described as to how it appears when it is printed, rather than how it is stored by the computer.

statement. A syntactically valid combination of words and symbols, beginning with a verb, that is written in the Procedure Division. A statement combines COBOL reserved words and user-defined operands.

subfile. A group of records of the same record format that can be displayed concurrently at a work station. The system sends the entire group of records to the work station in a single operation and receives the group in another operation.

subject of entry. A data-name or reserved word that appears immediately after a level indicator or level-number in a Data Division entry. It serves to reference the entry.

subprogram. A called program.

subscript. A positive integer whose value refers to a particular element in a table.

subscripted data-name. A data-name that has been made unique through the use of a subscript.

switch-status condition. A condition that states that a switch is currently on or off.

Synchronous Data Link Control. A discipline conforming to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute (ANSI) and High-level Data Link Control (HDLC) of the International Standards Organization (ISO), for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges may be duplex or half-duplex over switched or nonswitched links. The configuration of the link connection may be point-to-point, multipoint, or loop. Abbreviated SDLC.

system-name. An IBM-defined name that has a predefined meaning to the COBOL compiler. System-names include computer-names, language-names, and implementor-names.

table. A set of logically consecutive data items that are defined in the Data Division by means of the OCCURS clause.

table element. A data item that can be referenced in a table.

test condition. A statement that, taken as a whole, may be either true or false, depending on the circumstances existing at the time the expression is evaluated.

text-name. A user-defined word that identifies library text.

text-word. Any character-string or separator, except the space, in copied COBOL source or in pseudo-text.

TRANSACTION file. An input/output file used to communicate with work stations and/or for intersystems communications.

unary operator. A plus sign (+) 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.

UPSI switch. A program switch that performs the functions of a hardware switch. Eight switches are provided: UPSI-0 through UPSI-7.

user-defined word. A word, required by a clause or a statement, that must be supplied by the user in a clause or statement.

user-name. A type of implementor-name that appears in the VALUE OF clause, and that follows the rules for the formation of a user-defined word.

variable. A named modifiable value. The value can be accessed or modified by referring to the name of the variable.

verb. A COBOL reserved word that expresses an action to be taken by a COBOL compiler or an object program.

word. A character-string of not more than 30 characters, which forms a user-defined word, a system-name, or a reserved word.

work station. A device that lets a person transmit information to or receive information from a computer as needed to perform his job.

WORKING-STORAGE SECTION. A section-name (and the section itself) in the Data Division. The section describes records and noncontiguous data items that are not part of external files but are developed and processed internally. It also defines data items whose values are assigned in the source program.

zoned decimal format. Representation of a decimal value by 1 byte per digit. Bits 0 through 3 of the rightmost byte represent the sign; bits 0 through 3 of all other bytes represent the zone portion; bits 4 through 7 of all bytes represent the numeric portion. For example, in zoned decimal

format, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Contrast with *packed decimal format*.

zoned decimal item. A numeric data item that is represented internally in zoned decimal format.

Index

A

- abbreviated combined relation condition
 - description 5-19
 - examples 5-19
- ACCEPT statement
 - data transfer 5-30
 - DATE 5-33
 - DAY 5-33
 - description 5-30
 - for TRANSACTION file attributes 7-44
 - formats 5-31
 - mnemonic-name in 3-7
 - system information transfer 5-33
 - TIME 5-33
- ACCESS IS DYNAMIC
 - relative key required 3-24
 - WRITE statement 5-74
- ACCESS IS RANDOM
 - relative key required 3-24
 - WRITE statement 5-74
- ACCESS IS SEQUENTIAL
 - relative key optional with 3-24
 - WRITE statement 5-74
- ACCESS MODE clause
 - default is SEQUENTIAL 3-23
 - description 3-23
 - formats 3-17
- access modes
 - access modes 3-15
 - compiler-directing statement 5-4
 - file 7-41
 - FORMAT phrase, for TRANSACTION file 7-37
 - logical file 9-31
- access path
 - arrival sequence 9-15
 - example for indexed files 9-37
 - file processing considerations 9-40
 - for indexed files 1-2, 3-23
 - keyed sequence 9-15
- acknowledgment xv
- acquire program device
 - See ACQUIRE statement
- ACQUIRE statement
 - description 7-45
 - format 7-45
- actual decimal point
 - specification 4-50
- ADD statement
 - common phrases 5-82
 - composite of operands 5-80
 - description 5-84
 - formats 5-84
- ADVANCING phrase
 - of WRITE statement 5-72
- AFTER ADVANCING phrase
 - of WRITE statement 5-72
- AFTER phrase of INSPECT statement 5-95
- algebraic comparison
 - relation condition 5-11
 - sign test uses 5-14
- algebraic sign, definition 4-22
- alias name 6-40
 - example 9-19
- alignment rules
 - alphabetic items 4-21
 - alphanumeric edited items 4-21
 - alphanumeric items 4-21
 - decimal point in arithmetic statements 4-21
 - in an elementary MOVE statement 5-100
 - JUSTIFIED clause modifies 4-37
 - numeric edited items 4-21
 - numeric items 4-21
- ALL literal figurative constant
 - description 2-7
- ALL phrase of INSPECT statement 5-91
- ALL PROCEDURES phrase (DEBUGGING) 6-68
- allocate object command 9-5
- alphabet-name
 - CODE-SET clause specification 4-16
 - formation rules 2-5
- alphabet-name clause
 - COLLATING SEQUENCE phrase and 3-9
 - description 3-9
 - format 3-5
 - literal phrase 3-10
 - NATIVE phrase 3-9
 - PROGRAM COLLATING SEQUENCE clause and 3-9
 - STANDARD-1 phrase 3-10
- alphabetic characters
 - COBOL character set 2-3
 - description 2-3
 - in CURRENCY SIGN clause 3-12
- ALPHABETIC class test rules 5-9
- alphabetic item
 - alignment rules 4-21
 - PICTURE clause considerations 4-46
- alphanumeric edited item
 - alignment rules 4-21
 - PICTURE clause considerations 4-47
- alphanumeric item
 - JUSTIFIED clause and 4-37
 - PICTURE clause considerations 4-47
 - RECORD KEY data item 3-25
 - status key 3-26
- ALSO phrase of alphabet-name clause 3-10
- ALTER statement
 - description 5-116
 - format 5-116
 - segmentation considerations 5-117, 6-53
- altered GO TO statement 5-117, 5-119

- American National Standard COBOL
 - extensions to, printing of 1-5
 - publications xv
- AND logical connective
 - description 2-6
 - in combined condition 5-15
- AND NOT as logical connective 2-6
- apostrophe
 - specified in PROCESS statement 8-9
 - used as quotes 2-3
 - within nonnumeric literal 2-9
- Arabic numeral
 - description 2-4
 - in COBOL character set 2-3
- Area A, columns 8 through 11 2-12
- Area B, columns 12 through 72 2-12
- arithmetic expression
 - COMPUTE statement operand 5-85
 - description 5-6
 - in relation condition 5-11
 - in sign test 5-14
 - in WHEN phrase of SEARCH ALL 6-18
 - operators used 5-7
- arithmetic operation order rules 5-7
- arithmetic operations, combining 5-86
- arithmetic operator
 - description 5-7
 - list of 2-3
- arithmetic statement operands
 - overlapping 5-81
 - size of 5-80
- arithmetic statements
 - ADD statement 5-84
 - common phrases 5-82
 - COMPUTE statement 5-85
 - CORRESPONDING phrase 5-82
 - DIVIDE statement 5-86
 - GIVING phrase 5-83
 - multiple results 5-81
 - MULTIPLY statement 5-88
 - operands 5-80
 - ROUNDED phrase 5-83
 - SUBTRACT statement 5-89
- arithmetic symbol pair list 5-8
- ASCENDING/DESCENDING KEY phrase
 - of OCCURS clause
 - description 6-12
 - formats 6-10
 - SORT/MERGE
 - description 6-30
 - length of KEY data item 6-31
- ASCII
 - alphabet-name clause and 3-10
 - COLLATING SEQUENCE phrase and 6-31
 - collating sequences H-1
 - PROGRAM COLLATING SEQUENCE
 - clause 3-10
- ASSIGN clause
 - description 3-19
 - formats 3-17
 - indicators 7-6
- assigning index values 6-22
- assignment-name
 - as function-name 2-5
- ASSIGN clause
 - association 3-21
 - attribute 3-19
 - description 3-19
 - device 3-19
 - formats 3-16
 - hopper 3-21
 - name 3-21
 - RERUN clause 3-28
- associated card files D-1
- assumed decimal point
 - alignment in numeric item 4-21
 - description 4-21
- asterisk (*)
 - begins comment line 2-16
 - precedes comment line 2-10
- AT END condition
 - and SEARCH ALL statement 6-18
 - EXCEPTION/ERROR Declarative and 5-21
 - READ statement considerations 5-54, 7-53, 7-57
- AT END phrase
 - of SEARCH statement 6-17
 - status key 5-28
- attribute data for program device
 - formats I-10
 - obtaining 7-45
- ATTRIBUTE-DATA mnemonic-name
 - and ACCEPT statement 7-45
 - formats I-10
- AUTHOR paragraph, Syntax Checker
 - restriction 8-2
- availability of records 5-29

B

- BEFORE ADVANCING phrase
 - WRITE statement 5-72
- BEFORE/AFTER phrase of INSPECT
 - statement 5-95
- binary item
 - USAGE clause considerations 4-33
- binary operators 5-7
- bit configuration of hexadecimal digits 4-34
- blank line
 - description 2-16
- BLANK WHEN ZERO clause
 - description 4-37
 - format 4-37
 - VALUE clause considerations 4-40
- BLOCK CONTAINS clause
 - description 4-10
 - format 4-10
 - I-O-FEEDBACK special register and 4-11

blocking output record
 See unblocking input records/blocking output records

blocking, automatic 5-27

Boolean data facilities
 See also indicators
 comparison rules 5-13
 description 7-4, 7-44
 format 4-25, 7-6
 sending/receiving items 5-100

Boolean literal delimiters (B" and ")
 placement rules for 2-15

bottom page margin in LINAGE clause 4-13, 4-14

boundary alignment 4-36

braces indicate required items 1-4

brackets indicate optional items 1-5

breakpoints
 and COBOL functions A-7
 considerations 10-8
 data-name considerations 10-7
 description 10-4
 example 10-4
 names and COBOL functions J-7

BSC file support 7-1

C

CALL GDDM
 See graphics support

CALL statement
 description 6-59
 dynamic 6-59
 examples 6-64
 formats 6-59
 inter-program communication concepts 6-59
 ON OVERFLOW phrase 6-59
 segmentation considerations 6-53
 static 6-59
 USING phrase 6-60

called program
 segmentation considerations 6-53

calling for HELP 11-5

calling program
 segmentation considerations 6-53

CANCEL statement
 description 6-62
 example 6-63
 format 6-62
 inter-program communication concepts 9-60

capital letters, reserved words 1-4

card files, associated D-1

categories of data, concepts 4-20

categories of statements 5-4

character codes and CODE-SET clause 4-16

character set, COBOL definition 2-3

character-string
 and item size 4-22
 description 2-3

detailed description 2-4
 in INSPECT statement 5-91
 representation in PICTURE clause 4-44

characters allowed
 COBOL program 2-3
 nonnumeric literal 2-9
 numeric literal 2-9
 user-defined word 2-4

CHARACTERS phrase of BLOCK CONTAINS
 clause 4-10

CHARACTERS phrase of INSPECT statement 5-96

characters used in PICTURE clause 4-42

class condition
 description 5-9
 EBCDIC signs in 5-9
 format 5-9

class test rules 5-9

classes of data, concepts 4-20

clause, description 2-2

clauses, summary J-1

CLOSE statement
 access considerations 5-36
 description 5-35
 device considerations 5-36
 FOR REMOVAL phrase 5-35
 for TRANSACTION file 7-46
 formats 5-35, 7-46
 LOCK phrase 5-35
 organization considerations 5-36
 REEL/UNIT phrase 5-35
 volume considerations 5-36

COBOL character set
 description 2-3

COBOL coding form example 2-12

COBOL command statement
 compiler options specified in 8-6
 options 8-8
 syntax 8-6
 used to compile a COBOL program 8-6

COBOL definitions
 clause 2-2
 paragraph 2-2
 section 2-2
 statement 2-2

COBOL program structure
 general description 2-1

COBOL reserved words G-1

COBOL syntax checker
 restrictions on source 8-2
 used by SEU 8-2

COBOL words, detailed description 2-4

COBOL, industry standards xiv

CODE-SET clause
 description 4-16
 format 4-16
 omission of 4-16
 specified for diskette files 4-16
 specified for tape files 4-16

coding example
 COPY DDS results 6-45, 9-18

- Data Division 4-3
- DDS for a record format 9-17
- DDS for data base file 9-12
- DDS for field reference file 9-13
- DDS for keyed access path 9-37
- initialize a table to zero 6-8
- INSPECT statement 5-93
- PERFORM statement 5-124, 5-128, 5-132
- Procedure Division 5-6
- SEARCH statement 6-20
- SPECIAL-NAMES paragraph 3-9
- subscripting 6-5
- COLLATING SEQUENCE phrase
 - alphabet-name clause and 3-9
 - of SORT/MERGE statements 6-32
- collating sequences
 - EBCDIC and ASCII H-1
 - user-specified 3-10
- column 7
 - continuation area 2-12
 - D denotes debugging line 6-72
- columns 1 through 6 for sequence numbers 2-11
- combined arithmetic operations 5-86
- combined condition
 - description 5-16
 - format 5-16
- comma (,)
 - character, description 2-4
 - in Configuration Section 3-4
 - in data description entry 4-26, 4-27
 - in File-Control entry 3-18
 - in I-O-CONTROL paragraph 3-28
 - separator, rules for using 2-11
 - series connective 2-6
- comma and decimal point, interchanging 3-12
- comment
 - detailed description 2-10
 - punctuation characters valid in 2-16
- comment character-string, definition 2-3
- comment-entry
 - as a comment 2-10
 - detailed description 2-16
 - in Identification Division 3-1
- COMMIT statement
 - description 5-39
 - format 5-39
- commitment control
 - considerations 9-44
 - recovery after failure using 9-55
 - sample program 9-47
- COMMITMENT CONTROL clause
 - description 3-29
 - format 3-27
- common data concepts 4-17
- common key 9-15
- common phrases, arithmetic statements
 - CORRESPONDING phrase 5-82
 - GIVING phrase 5-83
 - ROUNDED phrase 5-83
 - SIZE ERROR phrase 5-84
- common processing facilities
 - current record pointer 5-29
 - INTO/FROM phrases 5-28
 - invalid key condition 5-28
 - status key 5-28
- communications file support 7-1
- communications recovery 9-55
 - sample program 9-57
- comparison rules
 - Boolean operands 5-13
 - INSPECT statement 5-92
 - START statement 5-66
- compilation
 - with a remote System/36 file 3-13
 - WITH DEBUGGING MODE 6-66
- compilation date in source listing 3-3
- compiler
 - data areas A-3
 - debugging examples A-6
 - debugging options A-5
 - IRP layout A-9
 - overview A-1
 - phases A-3
 - service information A-1
 - subroutines A-3
 - temporary result field 5-82
- compiler action on intermediate results E-1
- compiler features 1-3
- compiler messages C-1
- compiler options
 - description 8-8
 - specified on CRTCLPGM command 8-6
 - specified on PROCESS statement 8-13
- compiler output
 - listing descriptions 8-16
 - listing examples 8-17
- compiler-directing statement
- compiling source programs 8-5
- complex conditions
 - combined conditions 5-16
 - in PERFORM statement 5-123
 - negated simple conditions 5-15
- composite of operands
 - ADD statement execution and 5-84
 - arithmetic statements 5-80
 - description 5-80
 - SUBTRACT statement execution rules 5-89
- COMPUTATIONAL item
 - USAGE clause considerations 4-32
- COMPUTE statement
 - description 5-85
 - format 5-86
- computer-name
 - as system-name 2-5
 - form of 3-6
- concatenating data items 5-104
- concepts
 - data description 4-17
 - Sort/Merge 6-25
- concepts, inter-program communication

- common data 6-55
- control transfers 6-55
- language considerations 6-56
- concepts, segmentation
 - control 6-52
 - fixed segments 6-50
 - independent segments 6-51
 - logic 6-51
- condition
 - complex 5-15
 - in IF statement 5-24
 - permissible element sequences 5-17
 - simple 5-9
- condition-name
 - and SET statement 5-103
 - condition 5-10
 - description 4-26
 - formation rules 2-5
 - qualification format 2-19
 - switch-status condition 5-14
 - VALUE clause considerations 4-38
- condition-name condition
 - description 5-10
 - example 5-10
 - format 5-10
 - PROGRAM COLLATING SEQUENCE
 - clause 3-6
- condition-name entry
 - concepts 4-20
 - description 4-40
 - general format 4-24
- conditional expressions
 - complex conditions 5-15
 - evaluation rules 5-16
 - in PERFORM statement 5-122
 - simple conditions 5-9
- conditional GO TO statement 5-119
- conditional PERFORM statement 5-122
- conditional statement
 - categories of 5-4
 - description 5-4
 - IF statement
 - description 5-24
 - format 5-24
 - nested IF statement 5-25
- conditional variable
 - condition-name condition tests 5-10
 - condition-name entries 4-40
 - description 4-26
 - FILLER allowed as name 4-25
- Configuration Section
 - description 3-4
 - format 3-5
- connective words, detailed description 2-6
- considerations, system dependent
 - DATA DIVISION considerations
 - BLOCK CONTAINS clause 4-10
 - COPY DDS statement 6-40
 - index literals 6-7
 - item size 4-10
 - LINAGE clause 4-14
 - OCCURS clause 6-9
 - RECORD CONTAINS clause 4-11
 - SORT/MERGE statement 6-30
 - subscript literals 6-4
 - ENVIRONMENT DIVISION considerations
 - ASSIGN clause 3-19
 - RECORD KEY clause 3-24
 - RESERVE clause 3-21
 - SAME AREA or SAME RECORD AREA
 - clause 3-28
 - SAME SORT-MERGE AREA clause 6-27
 - general considerations
 - indexed file 3-23
 - library-name 2-19
 - program-name 3-2
 - relative file 3-23
 - source program library 6-39
 - source statements 8-2
 - text-name 2-19
 - user-defined words 2-4
 - PROCEDURE DIVISION considerations
 - arithmetic statements 5-80
 - CALL statement 6-59
 - GO TO DEPENDING ON statement 5-118
 - INSPECT statement 5-90
 - STOP statement 5-133
 - UNSTRING statement 5-109
- contents of DEBUG-ITEM special register 6-70
- continuation area
 - column 7 2-12
 - D denotes debugging line 6-72
- continuation line, definition 2-15
- control flow
 - PERFORM statement 5-122
 - SEARCH ALL statement 6-18
 - SEARCH statement 6-15
- control of segmentation 6-52
- control return, in PERFORM statement 5-121
- control transfer
 - changed by ALTER statement 5-116
 - inter-program communication concepts 6-55
 - PERFORM statement 5-121
- control transfer rules
 - Declarative procedures 5-20
 - explicit, GO TO statement 5-118
- control transfers, explicit and implicit 2-22
- CONTROL-AREA clause 7-42
- conversion of data
 - DISPLAY statement and 5-43
- COPY DDS, use with indicators 7-7, 7-49
- COPY statement
 - and externally described data 6-40
 - and EXTERNALLY-DESCRIBED-KEY 6-40
 - and floating point 6-43
 - and record description entry 4-5
 - data field structures 6-43
 - DDS and use of 6-40
 - DDS results 6-44, 9-18
 - description 6-39
 - example 6-49

- format 6-39
- phrases 6-39
- REPLACING phrase 6-47
- use with TRANSACTION files 7-1
- COPY, within PROCESS statement 8-15
- CORRESPONDING phrase
 - description 5-82, 5-103
 - FILLER items ignored 4-28
 - MOVE statement considerations 5-97
- count field in INSPECT statement 5-91
- COUNT IN phrase of UNSTRING statement 5-111
- CPF graphics support
 - See graphics support
- CR (credit) PICTURE symbol
 - description 4-44
 - sign control symbol 4-50
- creating a program 8-1
- CRTCLPGM command 8-6
- currency sign
 - description 2-3
 - fixed insertion symbol 4-50
 - floating insertion symbol 4-51
- CURRENCY SIGN clause
 - description 3-12
 - format 3-5
 - in PICTURE character-string 4-43
 - valid characters 3-12
- current record pointer
 - description 5-29
 - START statement 5-66

D

- data alignment
 - in an elementary MOVE statement 5-100
 - nonnumeric items 4-22
 - numeric items 4-21
- data attribute specification 2-21
- data base support
 - file description entry 4-6
 - logical record 4-1
 - physical record 4-1
- data categories
 - PICTURE clause 4-46
- data classes, description 4-20
- data communications file 7-1, 7-39
- data conversion
 - DISPLAY statement 5-43
 - in an elementary MOVE statement 5-99
 - numeric items 4-46
 - SET statement 6-23
- data description
 - arithmetic statement operands 5-82
- data description entry
 - general description 4-23
 - general formats 4-23
- data description specifications (DDS)
 - and externally described files 9-8

- and FORMATFILE files 9-22
- and multiple device files 7-30
- and program described files 9-20
- example for data base file 9-12
- example for field reference file 9-13
- example for keyed access path 9-37
- example for record format 9-17
- use of keywords 9-11
- Data Division
 - concepts 4-1
 - data description 4-17
 - entries 2-14
 - example 4-3
 - file description entry 4-5, 4-6
 - general description 2-1
 - inter-program communications concepts 6-56
 - organization
 - description 4-2
 - format 4-24
 - punctuation in 2-17
 - sort/merge considerations 6-28
 - table handling considerations
 - OCCURS clause 6-9
 - USAGE IS INDEX clause 6-13
- data hierachies
 - concepts 4-17
 - used in qualification 2-17
- data item
 - description entry concepts 4-17
 - figurative constant length 2-8
- data item description entry
 - ADD statement considerations 5-84
 - breaking apart 5-109
 - concatenating 5-104
 - general description 4-23
 - general format 4-3
 - joining together 5-104
 - MOVE statement considerations 5-98
 - subject of OCCURS clause 6-10
 - SUBTRACT statement considerations 5-89
- data manipulation statements
 - INSPECT statement 5-90
 - MOVE statement 5-97
 - STRING statement 5-104
 - UNSTRING statement 5-109
- data organization, description 3-14
- data receiving fields (UNSTRING) 5-110
- data record size specification 4-13
- data records
 - in file on another system 3-13
- DATA RECORDS clause
 - description 4-13
 - format 4-13
- data reference, methods of 2-17
- data references in Procedure Division 2-22
- data relationships 4-2
- data transfer
 - ACCEPT statement 5-30
 - DISPLAY statement 5-43
 - STRING statement 5-104

- UNSTRING statement 5-109
- data truncation
 - ACCEPT statement 5-30
 - incompatible record lengths 4-11
 - nonnumeric items 4-22
 - numeric items 4-21
- data-count fields in UNSTRING statement 5-110
- data-name
 - description 5-2
 - formation rules 2-5
 - qualification format 2-17
 - REDEFINES clause specification 4-28
 - restriction on duplications 2-20
 - subscript, definition 6-4
- data-name clause
 - description 4-25
 - format 4-27
 - order of specification 4-25
- date of compilation in source listing 3-3
- DATE-COMPILED paragraph
 - description 3-3
 - format 3-1
 - Syntax Checker restriction 8-4
- DATE-WRITTEN paragraph
 - Syntax Checker restriction 8-4
- DATE, ACCEPT statement 5-33
- DAY, ACCEPT statement 5-33
- DB (debit) PICTURE symbol
 - and numeric edited items 4-48
 - description 4-44
 - sign control symbol 4-50
- DB-FORMAT-NAME
 - other files 5-30
 - TRANSACTION files 7-35
- DB-FORMAT-NAME special register 5-30
- DD name 6-40, 9-8
- DDM files 3-13
 - definition GLOSS-4
- DDS name 6-40, 9-8
- DEBUG module, 1974 Standard 1-2
- DEBUG-CONTENTS 6-71
- DEBUG-ITEM special register
 - description 6-70
 - format 6-70
 - subfield contents 6-71
- debugging Declaratives, execution of 6-68
- debugging features
 - compile-time switch 6-66
 - execution-time switch 6-67, 10-12
 - USE FOR DEBUGGING procedures 6-68
- DEBUGGING MODE as compile-time switch 6-66
- decimal numbers, representation 4-22
- decimal point (.)
 - alignment of numeric items 4-21
 - alignment of numeric-edited items 4-21
 - and comma, interchanging 3-12
 - in elementary MOVE statement 5-100
 - in numeric literal 2-9
- DECIMAL-POINT IS COMMA clause
 - comma and period PICTURE symbols 4-44
 - description 3-12
 - format 3-5
- Declarative procedures
 - common exit point 5-117
 - debugging 6-68
 - MERGE statement 6-28
 - SORT statement 6-28
- Declaratives
 - EXCEPTION/ERROR 5-21
 - EXCEPTION/ERROR for TRANSACTION file 7-66
 - FOR DEBUGGING 6-68
 - general description 5-1
 - general format 5-20
 - section requirements when used 5-1
- DECLARATIVES keyword
 - begins Declaratives 5-1
 - begins in Area A 2-12, 2-15
- decrementing index-name values 6-24
- decrementing operands 5-123
- default attributes are implicit 2-21
- DELETE statement (input/output)
 - access considerations 5-40
 - description 5-40
 - device considerations 5-40
 - format 5-40
 - organization considerations 5-40
 - with duplicate keys 5-42
- DELIMITED BY ALL phrase (UNSTRING)
 - description 5-109
- DELIMITED BY phrase
 - and STRING statement execution 5-105
- delimiter
 - for Boolean literal 7-44
 - for pseudo-text 2-11
 - in INSPECT statement 5-95
 - in STRING statement 5-104
 - in UNSTRING statement 5-109
- DEPENDING ON phrase of GO TO statement
 - description 5-118
- DEPENDING ON phrase of OCCURS clause
 - description 6-11
 - format 6-10
- DESCENDING KEY phrase of OCCURS clause 6-12
- device dependencies 9-1
- device file
 - ASSIGN clause and 3-19
- diagnostic levels C-1
- diagnostic messages
 - description C-1
 - listing 8-24
 - severity levels 8-25
- direct and relative index usage 6-20
- direct indexing, description 6-7
- display device file
 - data description specifications for 7-1
 - description 7-39
 - record format 7-2
 - sample program 7-30
 - subfiles 7-22

- display file support 7-1
- DISPLAY phrase of USAGE clause
 - description 4-31
- display screen formats
 - COBOL coding form and 8-2
- DISPLAY statement
 - description 5-43
 - figurative constant length 2-8
 - format 5-43
 - mnemonic-name and 3-8
- distributed data management (DDM) 3-13
 - definition GLOSS-4
- DIVIDE statement
 - description 5-86
 - format 5-86
- division header 2-14
- division operator 5-7
- documentation, comments 2-10
- documenting end of procedures 5-117
- dollar sign (\$)
 - See currency sign
- dollar sign character, description 2-3
- DROP statement
 - description 7-47
 - format 7-47
- dump, formatted 10-1
- duplicate record keys, DUPLICATES phrase 3-24
- duplication of data-name, restriction 2-20
- dynamic access
 - DELETE statement 5-40
 - READ statement 5-54
 - WRITE statement 5-74
- dynamic access mode
 - description 3-15
 - in WHEN phrase of SEARCH ALL 6-18
 - indexed files 3-16
 - relative files 3-16
 - relative key required 3-24
- dynamic values in a table 6-7

E

- EBCDIC character set
 - COBOL characters 2-3
 - default for alphabet-name clause 3-9
 - NATIVE phrase 3-9
- EBCDIC collating sequence
 - alphabet-name clause 3-9
 - and HIGH-VALUE figurative constant 2-7
 - and LOW-VALUE figurative constant 2-7
 - and sort/merge phrase 6-31
 - list of characters H-2
- editing character 2-3
- editing in an elementary MOVE statement 5-99
- editing sign control symbols 4-50
- editing sign, description 4-22
- editing through PICTURE clause 4-49
- elementary item

- alignment rules 4-21
- as subscript 6-4
- classes and categories 4-20
- description 4-17
- level-number concepts 4-18
- MOVE statement operand 5-98
- valid clauses 4-26
- elementary moves
 - description 5-98
- ellipsis indicates repetition 1-5
- ELSE phrase
 - description 5-24
 - format 5-24
 - with nested IF statements 5-25
- END DECLARATIVES keywords
 - ends Declaratives 5-1
- end of execution
 - STOP RUN statement 5-133
- end of procedures, documenting 5-117
- ENDCBLDBG command 6-67
- ending file processing 5-35
- ENTCBLDBG command 6-67
- ENTER statement as documentation 5-134
- entering source programs 8-1
- entry, description 2-2
- Environment Division
 - Configuration Section 3-4
 - File-Control entry, sort/merge 6-27
 - File-Control paragraph 3-16
 - general description 2-1
 - I-O-Control entry, sort/merge 6-27
 - I-O-CONTROL paragraph 3-27
 - Input-Output Section 3-12
 - punctuation in 2-17
 - Sort/Merge considerations 6-27
 - SPECIAL-NAMES paragraph 3-7
- equal sign (=)
 - description 2-4
 - rules for using 2-11
 - separator, rules for using 2-10
- EQUAL TO relational operator
 - in WHEN phrase of SEARCH ALL 6-18
- error conditions
 - REWRITE statement considerations 5-63
- error correction, automatic 5-27
- evaluation results 5-17
- examples
 - See also sample programs
 - access path for indexed file 9-37
 - breakpoint 10-4
 - COBOL formatted dump 10-12
 - COBOL program skeleton coding 2-13
 - commitment control 9-44
 - compiler debugging options A-6
 - compiler options listing 8-16, 8-17
 - COPY DDS results 6-45, 9-18
 - COPY statement 6-49
 - cross reference listing 8-23
 - Data Division 4-9
 - Data Division coding 4-3

- Data Division map 8-20
- DDS for a display device file 7-1
- DDS for a record format 9-17
- DDS for a record format with Alias keyword 9-19
- DDS for data base file 9-12
- DDS for field reference file 9-13
- DDS for subfiles 7-25
- diagnostic messages listing 8-24
- Environment Division coding 3-4
- error recovery 9-55
- FIPS messages listing 8-22
- fixed insertion editing 4-51
- floating insertion editing 4-52
- FORMATFILE file 9-22
- generic START using a program described file 9-33
- generic START using an externally described file 9-33
- Identification Division coding 3-2
- Indicators 7-10
- initialize a table to zero 6-8
- INSPECT statement 5-93, 5-96
- inter-program communication 6-64
- mixed files 7-30
- multiple display files 7-30
- PERFORM statement 5-124, 5-128, 5-132
- Procedure Division 5-6
- Procedure Division coding 5-3
- record description concepts 4-18
- record format specifications 9-12, 9-13
- REDEFINES clause 4-29
- RENAMES clause 4-56
- ROLLING phrase 7-61
- SEARCH statement 6-20
- simple insertion editing 4-49
- source listing 8-17
- SPECIAL-NAMES paragraph 3-7
- STRING statement 5-106
- subscripting 6-4
- trace 10-9
- UNSTRING statement 5-114
- verb usage by count listing 8-20
- work station application programs 7-66
- zero suppression and replacement editing 4-53
- EXCEPTION/ERROR Declarative
 - description 5-21
 - EXTEND phrase 5-22
 - file-name phrase 5-21
 - format 5-21
 - I-O phrase 5-22
 - status key 5-28
- EXCEPTION/ERROR procedure
 - and USING phrase for sort/merge 6-32
 - CLOSE statement 5-35
 - DELETE statement 5-42
 - MERGE statement 6-28
 - REWRITE statement considerations 5-63
 - sort/merge OUTPUT PROCEDURE 6-32
- exceptional situations
 - SORT statement 6-28
- exceptions
 - and status key values 1-5
 - causes 9-52
- executing a program
 - compiler output 8-16
 - methods 8-25
- execution flow
 - ALTER statement 5-116
 - GO TO statement 5-118
 - PERFORM statement 5-120
 - SEARCH ALL statement 6-18
 - SEARCH statement 6-15, 6-18
 - STOP statement 5-133
- execution results
 - INSPECT statement examples 5-93, 5-96
 - JUSTIFIED clause examples 4-37
 - STRING statement 5-107
 - UNSTRING statement examples 5-114
- execution rules
 - INSPECT statement 5-92
 - PERFORM statement 5-122
 - ROUNDED phrase 5-83
 - SIZE ERROR phrase 5-84
 - STRING statement 5-105
 - UNSTRING statement 5-111
 - USE FOR DEBUGGING procedure 6-68
- execution sequence, performed procedures 5-122
- execution status, status key usage 3-27
- execution suspension
 - ACCEPT statement 5-30
 - STOP statement provides 5-133
- execution time
 - debugging switch 6-66
- exit point rules for performed procedures 5-120
- EXIT PROGRAM statement
 - CALL statement 6-56
 - description 6-63
 - format 6-63
 - inter-program communications concepts 6-64
- EXIT statement
 - description 5-117
 - format 5-118
- explicit attribute, description 2-21
- explicit control transfer, GO TO statement 5-118
- exponentiation operator 5-7
- exponentiation results 5-8
- EXTEND phrase of OPEN statement
 - description 5-50
- extensions, how printed 1-5
- extensions, summary of IBM B-1
- external data concepts 4-2
- external decimal item
 - See zoned decimal item
- externally described files
 - adding COBOL functions 9-18
 - coding record descriptions 9-10
 - considerations for using 9-8
 - description 9-8
 - overriding COBOL functions 9-18
- externally described TRANSACTION file 7-1

EXTERNALLY-DESCRIBED-KEY
and COPY statement, DDS, DD format 6-40,
6-45
description 3-26, 9-29

F

fall through of performed procedures 5-122
FD entry

See also File Description (FD) entry
description 4-1, 4-6

FILE-CONTROL paragraph 3-18

formats 4-7

implicit redefinition 4-28

LABEL RECORDS clause required 4-12

OPEN statement 5-48

field definitions

on remote system 3-13

when compiling programs 3-13

field-count field, in UNSTRING statement 5-110

fields, intermediate result E-1

figurative constant

detailed description 2-7

functions 2-7

file categories

data base files

logical 3-13

physical 3-13

device files 3-13

file creation time 5-73

File Description (FD) entry

description 4-6

FILE-CONTROL paragraph 3-18

formats 4-7

general description 4-1

general formats 4-7

LABEL RECORDS clause required 4-12

file feedback

See OPEN-FEEDBACK mnemonic-name

file label specification 4-12

file locking

allocate object command 9-5

lock states 9-5

shared files 9-5

file processing

access paths 9-40

associated card 9-30, D-1

card 9-29

DATABASE 9-31

DISK 9-31

feedback information 5-33

FORMATFILE 9-21

indexed organization 9-31

initiating 5-47

PRINTER 9-21

relative organization 9-38

sample programs F-1

sequential organization 9-39

summary 3-14

File Section

general description 4-5

general formats 4-5

VALUE clause considerations 4-38

FILE STATUS clause

CLOSE statement 5-35

DELETE statement 5-40

description 3-26

formats 3-18

INVALID KEY condition 5-28

READ statement 5-56

REWRITE statement 5-62

START statement 5-67

file status information

obtaining 10-12

related exceptions I-5

values I-5

file structure support summary I-1

File-Control entry

file processing entries 3-16

sort/merge considerations 6-26

TRANSACTION file processing entry 7-39

FILE-CONTROL paragraph

formats 3-16

function of 3-18

file-name

CLOSE statement operand 5-35

DELETE statement operand 5-40

formation rules 2-5

in FD entry 4-10

OPEN statement specification 5-48

READ statement considerations 5-53

SD entry operand 6-28

SELECT clause operand

description 3-19

formats 3-18

SORT statement operand 6-29

sort/merge file operand 6-27

START statement specification 5-67

file, description 4-1

files, TRANSACTION 7-1

FILLER keyword

description 4-27

order of specification 4-25

FIPS Flagger

compiler messages 8-22

description 6-73

standard modules used 1-3

1975 flagging 1-3

FIPS levels 1-4

FIRST phrase of INSPECT REPLACING
statement 5-93

FIRST phrase, READ statement 5-54

fixed insertion symbol

description 4-50

symbol

fixed length record

size specification 4-10

fixed length table

- description 6-11
- fixed page spacing, LINAGE clause 4-13
- fixed portion
 - segmented program 6-50
- floating insertion editing 4-51
- floating point fields 6-43
- footing area, LINAGE clause 4-14
- format notation, description 1-4
- FORMAT phrase, for TRANSACTION file 7-48
- FORMATFILE files
 - description 9-22
 - sample program 9-22
- formatted dump, COBOL
 - contents 10-12
 - example 10-13
- FROM identifier phrase
 - REWRITE statement considerations 5-63
 - WRITE statement considerations 5-75
- FROM phrase
 - ACCEPT statement 5-31
 - RELEASE statement 6-36
- function-name
 - ACCEPT statement 5-30
 - as system-name 2-5
 - DISPLAY statement operand 5-44
 - SPECIAL-NAMES paragraph 3-7
 - values 3-8
 - WRITE statement 5-76
- function-name-1 clause
 - description 3-7
 - format 3-5
- function-name-2 clause
 - description 3-8
 - format 3-5
 - switch-status condition and 3-8

G

- GDDM
 - See graphics support
- general description of S/38 COBOL 1-1
- generic START examples
 - using a program described file 9-33
 - using an externally described file 9-33
- GIVING phrase
 - arithmetic statements 5-83
 - SORT/MERGE statements 6-32
- GO TO statement 5-118
- Graphical Data Display Manager
 - See graphics support
- graphics support 6-61
- group moves 5-101

H

- hexadecimal digit bit configurations 4-34
- hyphen (-)
 - allowed in user-defined word 2-4
 - character, description 2-4
 - in continuation area, meaning 2-15
 - in program-name, conversion of 3-2
 - produced when copying Alias names 6-40

I

- I-O files
 - EXCEPTION/ERROR Declarative 5-21
- I-O option of OPEN statement
 - description 5-48
 - indexed file considerations 5-48
 - relative file considerations 5-48
 - TRANSACTION file considerations 7-48
- I-O-CONTROL paragraph
 - description 3-27
 - formats 3-27
 - order of clauses optional 3-28
 - sort/merge considerations 6-27
- I-O-FEEDBACK mnemonic-name and ACCEPT statement 5-33
 - description I-11
 - extended file status 7-41
- IBM extensions xii, 1-5, B-1
- Identification Division
 - description 3-1
 - format 3-1
 - punctuation in 2-16
- identifier
 - ACCEPT statement operand 5-30
 - breaking apart 5-109
 - description 5-2
 - DISPLAY statement operand 5-44
 - in sign test 5-14
 - INSPECT statement operand 5-90
 - replacing characters in 5-91
- IF statement
 - description 5-24
 - format 5-24
 - nested 5-25
- imperative-statement
 - categories of 5-4
 - description 5-4
- implicit attribute, description 2-21
- implicit control transfers 2-21
- IN as qualifier connective 2-6, 2-17
- incrementing index-name values 6-24
- incrementing operands
 - PERFORM VARYING rules 5-123
- indentation, to clarify logic 2-15
- independent segment

- calling and called programs 6-54
- description 6-51
- index
 - description 6-6
- INDEX usage
 - description 6-13
- index-name
 - and File Section 6-14
 - assigning values 6-22
 - comparison rules 6-13
 - description 6-6
 - in PERFORM statement 5-123
 - passing values of, in CALL 6-57
 - rules of formation 2-5, 6-13
 - SET statement operand 6-22
- index-name values 6-22
- INDEXED BY phrase
 - OCCURS clause
 - description 6-9
 - formats 6-9
 - SEARCH statement requirements 6-16
- indexed data item
 - comparison rules 6-14
 - description 6-13
- indexed file
 - File-Control entry
 - description 3-23
 - format 3-16
- INDEXED I-O module, 1974 Standard 1-2
- indexed organization 3-14
- indexes
 - assigning values 6-22
 - conditional variable 4-40
- indexing
 - description 6-6
 - INDEXED BY phrase rules 6-13
- INDICATOR clause 7-7
- indicators
 - and COPY statement 6-41, 7-7
 - and Separate indicator area (SI) attribute 7-5
 - associated with command keys 7-2
 - Boolean data items 7-4, 7-17
 - INDARA DDS keyword 7-5
 - sample programs 7-10
 - TRANSACTION file processing 7-4
- INDICATORS phrase 7-9, 7-49
- industry standards, COBOL xiv
- initialization
 - data items with INSPECT statement 5-91
 - DEBUG-ITEM special register 6-70
 - example for tables 6-8
 - indexed file considerations 5-48
 - LINAGE-COUNTER 4-15
 - of index 6-7
 - of table values 6-7
- input file
 - current record pointer used 5-29
 - for sort/merge 6-32
- INPUT phrase
 - EXCEPTION/ERROR Declarative 5-21
 - of OPEN statement
 - description 5-52
 - relative file considerations 5-52
- Input-Output Section
 - detailed description 3-12
 - format 3-13
- input/output errors
 - EXCEPTION/ERROR Declarative and 5-21
- INPUT/OUTPUT PROCEDURE control 6-35
- input/output statements
 - ACCEPT statement 5-31
 - ACQUIRE statement 7-45
 - CLOSE statement 5-35
 - common input/output phrases 5-27
 - DELETE statement 5-41
 - DISPLAY statement 5-44
 - DROP statement 7-47
 - OPEN statement 5-48
 - READ statement 5-52
 - REWRITE statement 5-63
 - START statement 5-66
 - WRITE statement 5-72
- insertion editing
 - description 4-49
- INSPECT statement
 - ALL literal figurative 2-7
 - BEFORE/AFTER phrase 5-95
 - coding example 5-93
 - comparisons illustration 5-92
 - description 5-90
 - examples 5-96
 - figurative constant length in 2-8
 - formats 5-90
 - REPLACING phrase 5-94
 - TALLYING phrase 5-94
- INSTALLATION paragraph as documentation
 - description 3-3
 - format 3-1
 - Syntax Checker restriction 8-4
- integer item
 - RECORD KEY data item 3-25
 - RELATIVE KEY data item 3-24
 - status key 3-26
- inter-program communication
 - common data 6-55
 - local data area 9-61
 - concepts
 - CALL statement 6-56
 - control transfers 6-55
 - language considerations 6-56
 - Data Division, Linkage Section 6-56
 - examples 6-64
 - EXIT PROGRAM statement 6-63, 9-61
 - CALL statement 6-64
 - file considerations 9-60
 - initialization 9-61
 - return of control 9-61
 - STOP RUN statement 6-64, 9-61
 - USING phrase, CALL statement 6-60

Inter-Program Communication module, 1974
 Standard 1-3
 interactive communications 7-1
 intermediate result fields E-1
 intermediate results
 SIZE ERROR phrase and 5-84
 internal data concepts 4-1
 internal decimal item
 See packed decimal item
 internal representation
 operational sign 4-22
 INTO identifier phrase of READ statement 5-56
 INTO phrase of RETURN statement 6-37
 INTO/FROM identifier phrase 5-28
 INVALID KEY condition
 actions taken 5-28
 EXCEPTION/ERROR Declarative and 5-22
 statements that recognize 5-28
 INVALID KEY phrase
 DELETE statement and 5-40
 START statement considerations 5-66
 status key 5-28
 WRITE statement 5-72
 IRP layout A-9

J

joining data items together 5-104
 JUSTIFIED clause
 description 4-36
 example of results 4-37
 format 4-36
 VALUE clause considerations 4-40

K

key fields
 common keys and 9-15
 defined by DDS 9-15
 descending keys 9-41
 for indexed files 9-32
 in the record area 5-41
 partial keys 9-33
 RECORD KEY clause 9-15, 9-32
 record keys and 9-15
 KEY phrase
 of OCCURS clause 6-12
 of START statement 5-67
 key, status 5-28
 keyword, detailed description 2-6
 keywords, printed as underlined capitals 1-4

L

label processing
 OPEN statement 5-51
 READ statement 5-59
 WRITE statement 5-79
 LABEL RECORDS clause
 description 4-12
 format 4-12
 required entry 4-12
 label specification 4-12
 language concepts, inter-program
 communication 6-54
 language extension
 notation used 1-4
 TRANSACTION file 7-1
 language structure, description 1-1
 language-name
 as system-name 2-5
 in ENTER statement 5-134
 LAST phrase, READ statement 5-54
 left parenthesis
 character description 2-3
 separator, rules for using 2-11
 length of figurative constant 2-8
 less than (<) character
 description 2-3
 when required in formats 1-5
 level check function
 externally described files 9-5
 level concepts 4-17
 level indicator
 as qualifier 2-19
 begins in Area A 2-13
 description 4-2
 level-66 entry 4-54
 level-number
 concepts
 description 4-18
 illustration 4-20
 description 4-2
 format 4-27
 formation rules 2-5
 REDEFINES specifications and 4-28
 unequal allowed 4-18
 01 and 77 begin in Area A 2-15
 02-49, 66, 88 begin in Area A or B 2-14
 level-01 item
 implicit redefinition 4-28
 level-01 records 4-18
 level-02 through -49 item 4-18
 level-66 entry
 description 4-54
 general description 4-26
 general format 4-24
 level-77 entry
 general description 4-25
 general format 4-24

- level-77 item
 - Linkage Section considerations 6-58
- level-88 entry
 - description 4-40
 - general format 4-24
- level-88 item
 - VALUE clause considerations 4-38, 4-40
- library
 - source program 6-39
 - test 10-1
- LIBRARY module 1-2
- library-name 2-5, 6-39
 - and System/38 library name 6-39
- LINAGE clause
 - description 4-13
 - format 4-14
 - LINAGE-COUNTER special register and 4-15
 - logical page depth illustrated 4-15
 - printer file commands 4-14
 - when assigned to PRINTER 9-21
 - with WRITE END-OF-PAGE 5-76
 - WRITE ADVANCING PAGE statement 5-76
- LINAGE-COUNTER special register
 - description 4-15
 - qualification rules 2-20
 - WRITE statement rules for 5-76
- line advancing
 - WRITE statement rules 5-76
- line continuation 2-15
- line-number, LINAGE-COUNTER value 4-15
- LINENUMBER 8-9, 8-14
- LINES AT BOTTOM phrase of LINAGE
 - clause 4-14
- LINES AT TOP phrase of LINAGE clause 4-14
- Linkage Section
 - general description 4-6
 - Inter-program Communication feature
 - concepts 6-56
 - description 6-56
 - level-77 and level-01 names unique 4-20
 - VALUE clause considerations 4-38
- literal
 - alphabet-name 2-5
 - as character-string 2-3
 - detailed description 2-8
 - in relation condition 5-13
 - INSPECT statement operand 5-90
 - phrase of alphabet-name clause 3-10
- local data area
 - and ACCEPT statement 5-30
 - and DISPLAY statement 5-43
 - description 9-65
 - LOCAL-DATA mnemonic-name 3-8
- LOCK phrase
 - CLOSE statement 5-36
- locking, file and record 9-5
- logic of segmentation 6-51
- logical connective 5-15
 - meaning and use 5-15
- logical connectives 2-6
 - detailed description 2-6
- logical file 3-13
- logical page positioning
 - LINAGE-COUNTER and 4-15
- logical page size
 - LINAGE clause specifies 4-13
- logical record
 - BLOCK CONTAINS CHARACTER, clause
 - and 4-11
 - description 4-1
 - level concepts 4-17
 - size specification 4-10
- loops in a program 9-54
- LOW-VALUE/LOW-VALUES figurative constant
 - description 2-7
- lower-case, user-defined words printed in 1-4

M

- manual
 - updates xii
- manual organization, description xi
- manual purpose xi
- margins of pages in LINAGE clause 4-14
- maximum length
 - COBOL word 2-4
 - data description entry 4-23
 - nonnumeric literal 2-9
 - numeric literal 2-9
 - of table 6-11
 - PICTURE character-string 4-42
 - Sort/Merge keys 6-31
 - table element 6-11
 - VALUE clause initialization 4-39
- maximum number
 - characters in numeric item 4-46
 - delimiters in UNSTRING statement 5-109
 - digits in numeric item 4-46
 - GO TO statement procedure-names 5-119
 - lines on printed page 4-13
 - Sort/Merge keys 6-30
- maximum value
 - of an index 6-7
 - subscript 6-4
- merge
 - concepts 6-27
 - description 6-27
- MERGE statement
 - description 6-29
 - format 6-29
 - phrases 6-30
 - segmentation considerations 6-54
 - sort/merge OUTPUT PROCEDURE 6-34
- merged records 6-29
- message reply modes 10-13
- messages, compiler
 - modifying C-2

- numbers C-1
- severity codes C-1
- methods of data reference 2-17
- MFCU, 5424 D-1
- minimum size
 - numeric item 4-47
- minimum value
 - index 6-7
 - subscript 6-4
- minus sign (-)
 - floating insertion symbol 4-51
 - in numeric literal 2-9
 - sign control symbol 4-50
- minus symbol (-) character, description 2-4
- minus symbol, when required in formats 1-5
- mixed files
 - description 7-40
 - sample program 7-30
 - subfiles 7-22
 - support 7-1
- mnemonic-name
 - as qualifier 3-9
 - formation rules 2-5
 - I-O-FEEDBACK 5-33
 - OPEN-FEEDBACK 5-33
- MOVE statement
 - CORRESPONDING phrase 5-97
 - description 5-97
 - elementary moves 5-98
 - formats 5-97
 - group moves 5-101
 - summary reference table 5-102
- MOVE statement, implicit
 - INTO/FROM identifier phrase 5-28
- multifunction card unit, 5424 D-1
- multiple device file
 - description 7-28
 - sample program 7-30
- MULTIPLE FILE TAPE clause
 - description 3-29
 - format 3-27
- multiple member processing 9-7
- multiple redefinitions allowed 4-29
- multiple results, arithmetic
 - description 5-81
 - execution rules 5-81
- multiplication operator 5-7
- MULTIPLY statement
 - description 5-88
 - formats 5-88

N

- NATIVE phrase
 - COLLATING SEQUENCE phrase 6-31
 - of alphabet-name clause 3-9
- negated simple condition
 - description 5-15

- format 5-15
- negative numeric data
 - SIGN clause and 4-35
- nested IF statement
 - description 5-25
 - examples 5-26
- next executable statement
 - description 2-22
- NEXT phrase of READ statement 5-54
- NEXT SENTENCE in IF statement 5-24
- NO DATA phrase
 - description 7-54
 - format 7-51
- NO REWIND phrase of CLOSE statement 5-36
- NO REWIND phrase of OPEN statement 5-48
- nonnumeric item
 - ALL literal figurative constant 2-7
 - HIGH-VALUE figurative constant 2-7
 - LOW-VALUE figurative constant 2-7
 - QUOTE figurative constant 2-7
 - SPACE, SPACES figurative constant 2-7
 - ZEROS figurative constant 2-7
- nonnumeric literal
 - alphabet-name clause 3-10
 - detailed description 2-9
 - punctuation characters 2-16
- NOT logical connective
 - meaning and use 5-15
 - placement in combined condition 5-16
- NUCLEUS module, 1974 Standard 1-2
- numerals, in COBOL character set 2-3
- numeric category, numeric literal 2-9
- numeric characters
 - allowed in user-defined word 2-4
 - description 2-4
 - list 2-4
- NUMERIC class test rules 5-9
- numeric edited item
 - alignment rules 4-21
- numeric edited item, PICTURE clause 4-48
- numeric first character in program-name 3-2
- numeric item
 - PICTURE clause considerations 4-47
 - ZERO, ZEROES, ZEROS figurative constant 2-7
- numeric literal
 - DECIMAL-POINT IS COMMA clause 3-12

O

- object of OCCURS DEPENDING ON clause
 - description 6-11
- object program
 - See program (executable)
- object time
 - See execution time
- OBJECT-COMPUTER paragraph
 - description 3-6

- format 3-5
- PROGRAM COLLATING SEQUENCE clause
 - relation conditions 3-6
 - SPECIAL-NAMES paragraph and 3-6
 - Syntax Checker restriction 8-4
- occurrence number
 - index-name 6-14
 - subscript identifiers 6-4
- OCCURS clause
 - ASCENDING/DESCENDING KEY phrase 6-12
 - DEPENDING ON phrase 6-11
 - description 6-9
 - fixed-length tables 6-11
 - formats 6-9
 - INDEXED BY phrase 6-13
 - variable-length tables 6-11
- omission of optional words allowed 2-6
- OMITTED phrase of LABEL RECORDS 4-12
- ON OVERFLOW phrase
 - and STRING statement execution 5-105
 - and UNSTRING execution 5-112
 - CALL statement 6-59
- one operand, varying 5-124
- OPEN INPUT statement
 - indexed file considerations 5-49
 - relative file considerations 5-49
- OPEN OUTPUT statement
 - LINAGE clause 4-14
- OPEN statement
 - access considerations 5-48
 - CLOSE statement 5-35
 - description 5-47
 - device considerations 5-48
 - for TRANSACTION file 7-47
 - formats 5-48
 - initializes LINAGE-COUNTER 4-15
 - organization considerations 5-48
 - sets current record pointer 5-29
- OPEN-FEEDBACK mnemonic-name
 - and ACCEPT statement 5-33
 - description I-11
- operand length
 - relational comparisons 5-11
- operands
 - overlapping 5-81
- operation order for arithmetic expressions 5-7
- operational sign
 - description 4-22
 - in an elementary MOVE statement 5-100
 - in class test 5-9
 - in numeric item 4-47
 - S PICTURE symbol specifies 4-42
 - SIGN clause 4-35
- operator response
 - ACCEPT statement 5-30
 - STOP statement 5-133
- OPTIONAL phrase of SELECT clause
 - description 3-19
 - format 3-17
- optional word

- detailed description 2-6
- OR as logical connective 2-6
- printed as capitals 1-4
- options, PROCESS statement 8-13
- OR condition, multiple UNSTRING 5-109
- OR NOT as logical connective 2-6
- order of clauses
 - I-O-CONTROL paragraph 3-27
- order of paragraphs, Identification Division 3-3
- order of symbols in PICTURE clause 4-45
- ordering records using sort/merge 6-25
- ORGANIZATION clause
 - default is SEQUENTIAL 3-22
 - formats 3-17
- Organization of Manual, description xi
- output device, DISPLAY statement 5-43
- output file
 - SAME clause and 3-28
- OUTPUT phrase
 - and data base files 5-49
 - EXCEPTION/ERROR Declarative 5-21
 - OPEN statement 5-48
- output procedure for sort/merge
 - description 6-34
- overflow condition
 - and UNSTRING execution 5-112
 - in a STRING statement 5-105
- overlapping delimiters in UNSTRING 5-109
- override file command 9-3

P

- packed decimal item
 - storage occupied 4-33
 - USAGE clause considerations 4-32
- padding of numeric-edited items 4-21
- padding with spaces
 - and DISPLAY statement 5-44
 - in a move 5-98
 - incompatible record lengths 4-11
 - nonnumeric items 4-22
- page advancing rules, WRITE statement 5-75
- page body, definition 4-14
- page ejecting 2-16
- page end, LINAGE clause specifies 5-76
- page margins in LINAGE clause 4-14
- page overflow
 - WRITE END-OF-PAGE considerations 5-77
- PAGE phrase of WRITE ADVANCING
 - statement 5-76
- page positioning
 - LINAGE-COUNTER 4-15
- page size, LINAGE clause specifies 4-13
- paragraph
 - description 5-2
- paragraph header, specification 2-14
- paragraph-names

- and PERFORM statement 5-121
- description 5-2
- formation rules 2-5
- GO TO statement 5-119
- qualification format 2-18
- restriction on duplication 2-19
- parentheses
 - separators, rules for using 2-11
- PERFORM statement 5-119
 - coding example 5-124, 5-128, 5-132
 - common exit point 5-121
 - conditional PERFORM 5-122
 - description 5-119
 - equivalent to sort/merge 6-35
 - for table search 6-20
 - formats 5-119
 - initializes index 6-7
 - range of 5-120
 - segmentation considerations 6-53
 - TIMES phrase 5-122
 - VARYING phrase 5-123
- performance considerations 9-53
- performed procedures
 - common exit point valid 5-117, 5-121
 - execution rules 5-120
- period (.)
 - character, definition 2-3
 - in Configuration Section 3-4
 - in data description entry 4-26, 4-27
 - in File-Control entry 3-18
 - in I-O-CONTROL paragraph 3-28
 - separator, rules for using 2-11
- permanent segment, definition 6-50
- permanent segments
 - ALTER statement 5-117
- permissible comparisons
 - relation-condition 5-11
- PGR
 - See graphics routines
- phrase, description 2-2
- physical file 3-13
- physical page size
 - logical page size 4-13
- physical record size
 - BLOCK CONTAINS clause 4-10
 - specifications 4-10
- physical record, definition 4-1
- PICTURE character-string
 - DECIMAL-POINT IS COMMA clause 3-12
 - description 2-4
 - item size 4-22
 - punctuation characters 2-10
- PICTURE clause
 - character-string representation 4-44
 - data categories 4-46
 - description 4-41
 - editing 4-49
 - editing sign function 4-22
 - fixed insertion editing 4-50
 - floating insertion editing 4-51
 - format 4-41
 - simple insertion editing 4-49
 - special insertion editing 4-50
 - symbol order 4-45
 - symbols used 4-42
 - VALUE clause considerations 4-39
 - zero suppression and replacement 4-52
- plural figurative constant 2-8
- plus sign
 - character definition 2-3
 - in numeric literal 2-9
 - SIGN clause 4-35
 - when required in formats 1-5
- POINTER phrase
 - and STRING statement execution 5-105
 - and UNSTRING execution 5-112
- positive data, and sign control symbols 4-51
- positive numeric data
 - SIGN clause 4-35
 - unsigned 4-22
- prerequisite publication xiii
- Presentation Graphics Routines
 - See graphics routines
- PRIOR phrase, READ statement 5-54
- problem determination 11-1
- procedure
 - Declarative
 - EXCEPTION/ERROR 5-21
 - for debugging 6-68
 - general description 5-20
 - general format 5-20
 - description 5-1
 - procedure branching statement
 - ALTER statement 5-116
 - GO TO statement 5-118
 - in IF statement 5-24
 - PERFORM statement 5-119
 - STOP statement 5-133
 - Procedure Division 5-1
 - arithmetic expressions 5-6
 - arithmetic statements 5-80
 - conditional expressions 5-8
 - conditional statements 5-23
 - data manipulation statements 5-90
 - data references 2-22
 - Declaratives 5-20
 - example 5-3
 - general description 2-1
 - input/output statements 5-27
 - LINAGE-COUNTER 4-15
 - organization 5-2
 - procedure branching statements 5-116, 5-118
 - punctuation 2-17
- procedure-name
 - ALTER statement operand 5-117
 - description 5-1
 - GO TO statement operand 5-118
 - PERFORM statement operand 5-122
- PROCESS statement
 - compiler options specified in 8-13
 - description 8-13

- format 8-13
- in batch compile environment 8-16
- options 8-13
- using COPY within 8-15
- processing a program 8-1
- processing associated card files D-2
- processing of files, initiating 5-47
- program (executable)
 - and table values 6-8
 - description 3-1
 - execution suspension (STOP) 5-133
- PROGRAM COLLATING SEQUENCE clause
 - alphabet-name clause 3-9
 - condition-name condition 3-6
 - description 3-6
 - format 3-5
 - relation condition 3-6
 - SPECIAL-NAMES paragraph 3-6
- program described files
 - considerations for using 9-8
 - description 9-8
 - externally defined by DDS with Create File commands 9-20
- program device
 - ACQUIRE operation 7-46
 - ACQUIRE operation failure 7-46
 - attributes of 7-46
 - invited 7-51
 - name
 - specification (see TERMINAL phrase)
 - used by last I-O operation (see CONTROL-AREA clause)
 - obtaining information about 7-51
 - release operation 7-47
 - status of 7-46
- program execution debugging switch 6-66
- program segments
 - description 6-50
 - fixed
 - permanent 6-50
 - independent 6-51
- program structure, general 2-1
- program switch
 - ALTER statement 5-117
- program syntax, debugging line 6-72
- program termination 8-26
- PROGRAM-ID paragraph
 - description 3-2
 - format 3-3
- program-name
 - description 3-2
 - formation rules 2-5
- pseudo-text
 - replacement rules 6-47
- pseudo-text delimiter
 - (=) separator, rules for using 2-11
 - placement rules for 2-11
- publication, prerequisite xiii
- publications, list of related xiii
- punctuation character

- defined as separator 2-3
- enclose nonnumeric literal 2-9
- list of 2-3
- with nonnumeric literal 2-9
- punctuation rules 2-16
- Purpose of Manual xi

Q

- qualification
 - CORRESPONDING phrase rules 5-82
 - detailed description 2-17
 - of UPSI condition-names 3-9
 - rules 2-19
- qualifier connectives 2-6
- qualifier, definition 2-17
- quotation mark
 - and QUOTE figurative constant 2-7
 - description 2-4
 - placement rules for 2-15
 - rules for using 2-11
 - specified in PROCESS statement 8-14
- QUOTE, QUOTES figurative constant 2-7
- quotient, in division 5-86

R

- random access
 - DELETE statement 5-40
 - description 3-15
 - indexed files 3-23
 - READ statement 5-52
 - relative files 3-23
 - WRITE statement 5-74
- read from invited program devices 7-51
 - controlled job termination 7-52
 - invited program device errors 7-51
 - name of program device read
 - See CONTROL-AREA clause
 - no data available
 - See NO DATA phrase
 - no program devices invited
 - See AT END phrase
 - operation 7-51
 - record format name read
 - See CONTROL-AREA clause
 - time out on wait for data 7-52
- READ statement
 - access considerations 5-54
 - description 5-52
 - device considerations 5-54
 - for TRANSACTION file 7-50
 - formats 5-52
 - INTO identifier phrase 5-28
 - organization considerations 5-54

- sets current record pointer 5-29
- receiving field
 - alignment rules and 4-21
 - in group MOVE statement 5-101
 - in STRING statement 5-105
 - in UNSTRING statement 5-111
 - MOVE statement 5-98
- record
 - See logical record
- RECORD CONTAINS clause
 - description 4-11
 - format 4-11
- record description entry
 - as RENAMES clause qualifier 4-54
 - COPY statement, DDS format 4-5
 - description 4-1, 4-23
 - sort/merge output file 6-34
- record format
 - fields 7-2
- record format specifications
 - example 9-12, 9-13
 - use of DDS keywords 9-11
- RECORD KEY
 - for a format 3-25
 - function of in indexed file 3-16
 - REWRITE statement 5-63
 - START statement 5-68
- RECORD KEY clause
 - description 3-24
 - format 3-18
- record level concepts 4-17
- record locking
 - updating data base records 9-5
- record sequencing using sort/merge 6-25
- record size
 - ACCEPT statement 5-30
 - DISPLAY statement 5-44
 - established at file creation time 5-73
 - incompatible lengths 4-11
 - RECORD CONTAINS clause specifies 4-11
 - REWRITE statement considerations 5-61
 - sort/merge output file considerations 6-34
- record storage, WRITE statement 5-73
- record-description level-number concepts
 - description 4-18
 - illustration 4-20
- record-name
 - formation rules 2-5
 - RELEASE statement operand 6-36
 - REWRITE statement 5-61
 - WRITE statement considerations 5-73
- RECORDS phrase of RERUN clause
 - format 3-27
- REDEFINES clause
 - description 4-28
 - format 4-28
- redefining item
 - description 4-29
 - moving to a redefined item 5-98
- reference number 8-17, 8-24
- reference to data 2-18
- register 6-70
- relation character
 - list of 2-3
- relation condition
 - description 5-11
 - format 5-11
 - nonnumeric operand comparisons 5-13
 - numeric operand comparisons 5-13
 - PROGRAM COLLATING SEQUENCE
 - clause 3-6
 - relational operator meanings 5-11
 - table handling rules 6-14
- relational operator
 - in abbreviated combined relation condition 5-19
 - meaning 5-11
- relative and direct index usage 6-20
- relative file organization, description 3-15
- relative files
 - File-Control entry
 - description 3-16
 - format 3-18
- RELATIVE I-O module, 1974 Standard 1-2
- relative indexing, description 6-7
- RELATIVE KEY
 - FILE-CONTROL paragraph
 - considerations 3-18
 - START statement 5-68
 - SUBFILE phrase considerations 7-49
 - WRITE statement considerations 5-75
- release program device
 - See DROP statement
- RELEASE statement
 - description 6-36
 - format 6-36
- REMAINDER phrase of DIVIDE statement
 - execution rules 5-87
 - format 5-87
- RENAMES clause
 - data-name-2 phrase 4-54
 - data-name-2 THRU data-name-3 phrase 4-54
 - description 4-54
 - format 4-54
 - general format 4-24
 - level-66 item 4-20
 - specification examples 4-56
- repetitive execution of PERFORM statement 5-122
- replacement editing
 - description 4-52
- replacement of file records 5-61
- replacement rules for library-text 6-39
- REPLACING phrase
 - of COPY statement 6-47
 - of INSPECT statement 5-94
- reporting COBOL problems 11-5
- required items indicated by braces 1-4
- required words, detailed description 2-6
- RERUN clause
 - description 3-28
 - formats 3-27
- RESERVE clause 3-17

- reserved word
 - detailed description 2-5
 - list G-1
 - printed as capital letters 1-4
- retrieving/saving source entries 6-39
- RETURN statement for sort/merge
 - description 6-36
 - format 6-37
 - sets current record pointer 5-29
- REVERSED phrase of OPEN statement 5-48
- REWRITE statement
 - access considerations 5-62
 - description 5-61
 - device considerations 5-62
 - for TRANSACTION file 7-58
 - format 5-61
 - FROM identifier phrase and 5-28
 - organization considerations 5-62
- right parenthesis ())
 - description 2-3
 - rules for using 2-11
- right-padding of items 4-21
- ROLLBACK statement
 - description 5-65
 - format 5-65
 - sets current record pointer 5-29
- ROUNDED phrase
 - ADD statement 5-84
 - COMPUTE statement 5-84
 - description 5-83
 - DIVIDE statement 5-84
 - execution rules 5-83
 - MULTIPLY statement 5-84
 - routine-name
 - formation rules 2-5
 - in ENTER statement 5-134
 - SUBTRACT statement 5-84
- rules for qualification 2-19
- rules, punctuation 2-16
- run unit
 - CALL statement transfer control 6-56

S

- SAME clause
 - and SAME SORT/SORT-MERGE AREA
 - clause 6-27
 - description 3-28
 - format 3-27
 - RELEASE statement and 6-36
- sample programs
 - See also examples
 - commitment control 9-47
 - error recovery procedure 9-55
 - FORMATFILE file creation 9-22
 - indexed file creation F-6
 - indexed file updating F-8
 - mixed file creation 7-30

- multiple display file creation 7-30
- relative file creation F-11
- relative file retrieval F-15
- relative file updating F-13
- sequential file creation F-2
- sequential file updating and extension F-4
- TRANSACTION file processing 7-50
- TRANSACTION program 7-67
- work station support 7-66
- SD entry
 - and MERGE statement file-name 6-29
 - and RELEASE statement record-name 6-36
 - and RETURN statement file-name 6-37
 - and SORT statement file-name 6-28
 - description 6-28
 - FILE-CONTROL paragraph 3-18
 - format 6-28
- SEARCH statement
 - coding example 6-20
 - description 6-15
 - execution considerations 6-18
 - formats 6-15
- section header
 - description 5-1
 - in Declarative procedures 5-20
 - specification of 2-14
- section-name
 - ALTER statement and 5-116
 - as qualifier 2-19
 - description 5-1
 - formation rules 2-5
 - GO TO statement 5-118
 - PERFORM statement 5-122
 - restriction on duplication 2-19
- section, description 5-1
- SECURITY paragraph
 - format 3-1
 - syntax checker restriction 8-4
- SEGMENT-LIMIT clause 3-6
- segment-number
 - description 6-50
 - formation rules 2-5
 - logic of specification 6-51
- segment-number, in Declaratives 5-20
- segmentation feature
 - concepts
 - program segments 6-50
 - Procedure Division 6-53
 - special considerations 6-53
- segmentation information
 - ALTER statement 6-53
 - calling and called programs 6-54
 - GO TO statement 6-53
 - PERFORM statement 6-54
 - SORT and MERGE statements 6-54
- SEGMENTATION module, 1974 Standard 1-2
- SELECT clause
 - formats 3-17
 - order of specification 3-19
 - sort/merge considerations 6-26

- sending field
 - in group MOVE statement 5-101
 - in STRING statement 5-105
 - in UNSTRING statement 5-109
 - MOVE statement 5-98
- sentence
 - categories 5-3
 - description 5-2
- SEPARATE CHARACTER phrase of SIGN clause
 - description and format 4-35
- separator
 - description 2-3
 - detailed description 2-10
- sequence number, detailed description 2-11
- sequence of clauses, required 1-5
- sequence of execution, performed procedures 5-122
- sequencing records using sort/merge 6-25
- sequential access
 - DELETE statement 5-40
 - READ statement and 5-52
 - relative key option for 3-24
 - WRITE statement 5-74
- sequential access mode
 - description 3-15
 - indexed files 3-23
 - relative files 3-23
 - sequential files 3-16
- sequential file
 - description 3-16
 - EXCEPTION/ERROR Declarative phrases 5-21
 - FILE-CONTROL Paragraph 3-16
 - format 3-16
 - OPEN statement considerations 5-48
 - organization 3-14
 - READ statement and 5-55
 - REWRITE statement 5-62
- SEQUENTIAL I-O module, 1974 Standard 1-2
- series connectives, detailed description 2-6
- service information, compiler A-1
- SET statement
 - and conditional variables 5-103
 - and external switches 5-103
 - and multidimensional table search 6-20
 - description 5-103, 6-22
 - formats 5-103, 6-22
 - initializes index 6-7
 - TO phrase 6-22
 - UP/DOWN BY phrase 6-24
 - valid field combinations 6-23
- SEU
 - See source entry utility (SEU)
- sharing storage
 - file records 3-27
- SI attribute 3-21, 7-5
- sign character, in numeric literal 2-9
- SIGN clause
 - description 4-35
 - format 4-35
 - operational sign representation 4-22
 - S PICTURE symbol 4-42
- sign condition
 - description 5-14
 - format 5-14
- sign control, fixed insertion editing 4-50
- sign in numeric literal, position of 2-9
- SIGN IS SEPARATE CHARACTER clause
 - description and format 4-35
- signed numeric item
 - SIGN clause specification 4-35
- simple condition
 - negation of 5-15
- simple insertion editing 4-49
- singular figurative constant equivalent to plural 2-8
- SIZE ERROR phrase
 - COMPUTE statement 5-84
 - description 5-84
 - DIVIDE statement 5-84
 - MULTIPLY statement 5-84
 - SUBTRACT statement 5-84
- size of DEBUG-CONTENTS 6-71
- size of operands in nonnumeric comparisons 5-13
- SIZE, STRING statement delimiter 5-104
- small letters, user-defined words printed in 1-4
- SORT statement
 - description 6-29
 - format 6-29
 - phrases 6-31
 - segmentation considerations 6-53
 - sort/merge OUTPUT procedure 6-35
- SORT-MERGE module, 1974 Standard 1-2
- Sort/Merge
 - concepts 6-25
 - considerations 6-37
 - Data Division – SD entry 6-28
 - Environment Division 6-27
 - File Description (SD) entry
 - description 6-28
 - FILE-CONTROL paragraph required for 3-18
 - format 6-28
 - File-Control entry 6-26
 - I-O-Control entry 6-27
 - or System/38 logical file support 6-25
 - Procedure Division
 - MERGE statement 6-29
 - RELEASE statement 6-36
 - RETURN statement 6-36
 - SORT statement 6-29
 - SORT/MERGE statement phrases 6-31
- source entry utility (SEU)
 - browsing a compiler listing 8-4
 - entering a source program 8-2
- source language debugging
 - compile-time switch 6-66
 - DEBUG-ITEM special register 6-70
 - debugging lines 6-72
 - execution-time switch 6-67
 - USE FOR DEBUGGING procedures 6-68
- source program
 - batch entry 8-1

- compiling 8-5
- COPY statement considerations 6-39
- description 3-1
- interactive entry 8-1
- using SEU to enter 8-1
- WITH DEBUGGING MODE switch and compilation 6-66
- source program library feature 6-39
- SOURCE-COMPUTER paragraph
 - DEBUGGING MODE as compile-time switch 6-66
 - format 3-4
 - Syntax Checker restriction 8-4
 - treated as documentation 3-6
- space fill
 - example using INSPECT 5-97
 - in an elementary MOVE statement 5-100
- space punctuation character, description 2-3
- space separator
 - rules for using 2-10
- SPACE/SPACES figurative constant
 - description 2-7
- spaces
 - BLANK WHEN ZERO clause causes insertion 4-37
- spacing
 - of pages and LINAGE clause 4-13
- special collating sequences, specifying 3-6
- special features
 - debugging 6-66
 - inter-program communication 6-54
 - segmentation 6-50
 - sort/merge 6-25
 - table handling 6-1
- special insertion editing 4-50
- special level-number concepts 4-20
- special registers
 - DATE 5-33
 - DAY 5-33
 - DB-FORMAT-NAME 5-30, 7-48
 - DEBUG-ITEM 6-70
 - need not be defined 2-6
 - TIME 5-33
 - use of 2-6
- SPECIAL-NAMES paragraph
 - ACCEPT statement 5-30
 - description
 - alphabet-name clause 3-11
 - CURRENCY SIGN clause 3-12
 - DECIMAL-POINT IS COMMA clause 3-12
 - function-name-1 clause 3-7
 - function-name-2 clause 3-8
 - example 3-9
 - format 3-5
 - PROGRAM COLLATING SEQUENCE clause 3-6
 - Syntax Checker restriction 8-4
- specification order
 - data description clauses 4-25
 - data-name or FILLER clause 4-25
 - level-number 4-25
 - REDEFINES 4-25
 - subscripts 6-5
- spooling files
 - inline data files 9-4
 - output files 9-4
- square brackets ([]) indicate optional items 1-5
- standard alignment rules
 - alphabetic items 4-21
 - alphanumeric items 4-21
 - JUSTIFIED clause modifies 4-37
 - numeric edited items 4-21
 - numeric items 4-21
- standard COBOL format
 - description 2-11
- standard data format
 - description 4-22
 - numeric literal size 2-9
- STANDARD phrase of LABEL RECORDS clause 4-12
- STANDARD-1 phrase
 - of alphabet-name clause 3-10
 - SEQUENCE phrase 6-31
- standards, COBOL xiv
- START statement
 - access considerations 5-67
 - description 5-66
 - device considerations 5-67
 - format 5-67
 - INVALID KEY phrases 5-68
 - organization considerations 5-67
 - relative key 3-24
 - sets current record pointer 5-29
- starting line number
 - duplicate record keys, DUPLICATES phrase 3-24
 - formula 7-61
 - WRITE statement, for TRANSACTION file 7-60
- statement
 - categories 5-4
 - description 5-2
 - summary J-1
- static values of a table 6-7
- status key values I-5
- STATUS KEY, file processing
 - description of use 5-28
 - OPEN statement 5-52
 - WRITE statement 5-73
- STOP RUN statement
 - description and format 5-133
 - inter-program communication 9-61
- STOP statement
 - ALL literal figurative constant restrictions 2-7
 - description 5-133
 - format 5-133
- storage allocation, calling and called programs 6-57
- storage format, USAGE clause specifies 4-31
- storage layout of table, example 6-2
- storage of records

- REDEFINES clause and 4-28
- storage of records illustrated 4-20
- STRING statement
 - ALL literal figurative constant restriction 2-7
 - description 5-104
 - examples 5-106
 - format 5-104
- structure of COBOL program, general
 - description 2-1
- subfield contents of DEBUG-ITEM special
 - register 6-71
- subfile
 - access 7-23, 7-50
 - special register DB-FORMAT-NAME 7-24
 - specified in DDS 7-22
 - use of 7-24
 - valid operations 7-23
- SUBFILE phrase 7-49
- subject
 - of abbreviated combined relation-condition 5-19
 - of OCCURS clause, definition 6-9
 - of relation condition
 - description 5-11
- subscripting
 - and PROCESS statement 6-5
 - description 6-4
 - example 6-5
 - invalid for File-Control entry data-names 3-18
 - restriction for qualifiers 2-20
- substitution field of INSPECT REPLACING 5-94
- SUBTRACT statement
 - common phrases 5-82
 - CORRESPONDING phrase 5-89
 - description 5-89
 - formats 5-89
- subtraction operator 5-7
- summary of clauses and statements J-1
- summary of updates xii
- suppression of sequence checking 2-11
- switch-status condition
 - description 5-14
 - format 5-14
- symbol order in PICTURE clause 4-45
- symbols used in PICTURE clause 4-42
- SYNCHRONIZED clause
 - description 4-36
 - format 4-36
- syntax of program
 - debugging lines 6-72
 - detailed description 2-3
- system console
 - ACCEPT statement 5-30
 - DISPLAY statement 5-44
- system information transfer, ACCEPT
 - DATE 5-33
 - DAY 5-33
 - TIME 5-33
- system input device, ACCEPT statement 5-32
- system-dependent considerations
 - DATA DIVISION considerations

- BLOCK CONTAINS clause 4-10
- COPY DDS statement 6-40
- index literals 6-7
- item size 4-10
- LINAGE clause 4-14
- OCCURS clause 6-9
- RECORD CONTAINS clause 4-11
- SORT/MERGE statement 6-30
 - subscript literals 6-4
- ENVIRONMENT DIVISION considerations
 - ASSIGN clause 3-19
 - RECORD KEY clause 3-24
 - RESERVE clause 3-21
 - SAME AREA or SAME RECORD AREA clause 3-28
 - SAME SORT-MERGE AREA clause 6-27
- general considerations
 - indexed file 3-23
 - library-name 2-19
 - program-name 3-2
 - relative file 3-23
 - source program library 6-39
 - source statements 8-2
 - text-name 2-19
 - user-defined words 2-4
- PROCEDURE DIVISION considerations
 - arithmetic statements 5-80
 - CALL statement 6-59
 - GO TO DEPENDING ON statement 5-118
 - INSPECT statement 5-90
 - STOP statement 5-133
 - UNSTRING statement 5-109
- system-error routine 5-22
- system-independent binary items 4-33
- system-name
 - description 2-5
- SYSTEM-SHUTDOWN as function-name 3-9
- System/36
 - accessing data files on remote 3-13
 - field definitions on remote 3-13
- System/38
 - accessing data files on remote 3-13
 - field definitions on remote 3-13
- SYSTEM/38 file name
 - in ASSIGN clause 3-21
 - in COPY statement 6-39
- SYSTEM/38 library name
 - and Library-name 6-39
 - in COPY statement 6-39

T

- table
 - definition 6-2
 - length 6-10
- table element
 - definition 6-1
 - length 6-10

- table handling
 - Data Division 6-9
 - OCCURS clause 6-9
 - Procedure Division 6-14
 - reinitializing index-names 6-22
 - relation conditions 6-14
 - SEARCH statement 6-15
 - SET statement 6-22
 - table definition 6-2
 - table initialization 6-7
 - table references 6-4
 - UP/DOWN BY phrase 6-24
 - USAGE IS INDEX clause 6-13
- table handling concepts
 - table definition 6-20
 - table initialization 6-7
 - table references
 - indexing 6-6
 - subscripting 6-4
- TABLE HANDLING module, 1974 Standard 1-2
- table layout, example 6-2
- table of valid and invalid moves 5-102
- table references
 - and SEARCH ALL results 6-20
 - indexing 6-6
 - subscripting 6-4
- table values, defining 6-7
- table, definition 6-2
- TALLYING phrase
 - INSPECT statement 5-94
 - UNSTRING statement 5-110
- TERMINAL phrase
 - description 7-49
 - with READ (nonsubfile), description 7-58
 - with READ statement, formats 7-50
 - with READ SUBFILE, description 7-56
 - with REWRITE statement, description 7-58
 - with REWRITE statement, format 7-58
 - with WRITE (nonsubfile), description 7-60
 - with WRITE statement (nonsubfile), format 7-60
 - with WRITE SUBFILE, description 7-63
 - with WRITE SUBFILE, format 7-63
- termination of execution
 - EXIT PROGRAM statement 6-63
 - STOP RUN statement 5-133
- termination, program 8-26
- test library 10-1
- testing function, CPF 10-1
- text-name
 - COPY statement operand 6-39
 - formation rules 2-5
 - qualification format 2-18
- THEN phrase
 - format 5-24
 - used as separator 5-24
- TIME, ACCEPT statement 5-30
- TIMES phrase of PERFORM statement 5-122
- TO phrase, SET statement 6-22
- top page margin in LINAGE clause 4-14
- traces
 - considerations 10-11
 - description 10-8
 - example 10-9
- TRAILING phrase of SIGN clause
 - description 4-35
- TRANSACTION files
 - Boolean data facilities 7-44
 - data description specifications for 7-1
 - Data Division Considerations 7-43
 - Environment Division considerations 7-39
 - externally described 7-1
 - language extensions for 7-1
 - Procedure Division considerations 7-44
 - processing externally described 7-4
 - program described 7-39
 - sample programs 7-66
- TRANSACTION organization 7-40
- transfer of control
 - ALTER statement change 5-117
 - and sort/merge OUTPUT PROCEDURE 6-35
 - explicit and implicit 2-22
 - sort INPUT PROCEDURE 6-33
- transfer of data
 - in a STRING statement 5-105
 - into DEBUG-ITEM special register 6-71
- truncation
 - in numeric items 4-21
 - JUSTIFIED clause 4-37
- truncation of data
 - ACCEPT statement 5-30
 - in an elementary MOVE statement 5-100
 - in floating insertion editing 4-52
 - incompatible record lengths 4-11
 - ROUNDED phrase 5-83
 - VALUE clause restrictions 4-39
- truth value, description 5-15
- twos complement form 4-34

U

- unary operator
 - list 5-7
 - use 5-7
- unblocked files, BLOCK CONTAINS clause 4-10
- unlocking input records/blocking output records
 - conditions 9-6
 - file status values 3-27
 - OPEN statement 5-52
 - updating the I-O-FEEDBACK area 1-11
- unlocking, automatic 5-27
- unconditional GO TO statement 5-119
- underlined capital letters, keywords as 1-4
- underscores, translated to hyphens 6-40
- unsigned numeric literal considered positive 2-9
- unsigned operand
 - considered positive or zero 4-22

- UNSTRING statement
 - data receiving fields 5-110
 - description 5-109
 - examples 5-114
 - execution rules 5-111
 - figurative constant length 2-8
 - format 5-109
 - sending field 5-109
- UNTIL phrase of PERFORM statement 5-122
- UP/DOWN phrase, SET statement 6-22
- updates to manual xii
- UPON phrase of DISPLAY statement 5-44
- UPSI switches
 - and SET statement 5-103
 - and switch-status condition 5-14
 - SPECIAL-NAMES paragraph 3-9
- UPSI-0 through UPSI-7 as function-names 3-8
- USAGE clause
 - and numeric items 4-32
 - computational options 4-32
 - description 4-31
 - DISPLAY phrase 4-31
 - format 4-31
 - operational sign representation 4-22
 - REDEFINES clause and data values 4-30
 - zoned decimal items 4-32
- USAGE IS INDEX clause
 - description 6-13
 - format 6-13
- USE statement
 - EXCEPTION/ERROR 5-21
 - EXCEPTION/ERROR for TRANSACTION file 7-66
 - FOR DEBUGGING 6-68
- user-defined word
 - detailed description 2-4
 - formation rules 2-4, 2-5
 - printed in lower case 1-4
- user-name
 - as function-name 2-5
 - in VALUE OF clause 4-13
- USING phrase
 - SORT/MERGE statement
 - description 6-32
- USING phrase, inter-program communication
 - format 6-60

V

- valid and invalid elementary move table 5-102
- valid characters in CURRENCY SIGN clause 3-12
- valid execution sequence, PERFORM statement 5-122
- validity checking, automatic 5-27

- VALUE clause
 - description 4-38
 - example of condition-name entries 4-41
 - format 4-38
- VALUE OF clause 4-12
- value, of numeric literal 2-9
- variable length table
 - description 6-11
 - format 6-10
- varying operands in PERFORM statement 5-123
- VARYING phrase
 - PERFORM statement 5-123
 - SEARCH statement 6-15
- verbs
 - as keyword 2-6
 - lists 5-5

W

- WHEN phrase of SEARCH ALL statement 6-18
- WITH FOOTING phrase of LINAGE clause
 - description 4-14
- WITH NO REWIND phrase of CLOSE 5-36
- word
 - description 2-3
 - detailed description 2-4
 - reserved, detailed description 2-5
- words, reserved G-1
- work station support
 - See also TRANSACTION files
 - general description 7-1
- Working-Storage Section
 - general description 4-5
 - general format 4-3
 - level-77 and level-01 names unique 4-20
 - VALUE clause considerations 4-38
- WRITE ADVANCING statement
 - description 5-75
 - LINAGE clause and 4-15
- WRITE statement
 - access considerations 5-74
 - ADVANCING phrase 5-74
 - description 5-72
 - device considerations 5-74
 - END-OF-PAGE phrase 5-74
 - for TRANSACTION file 7-60
 - format 5-72
 - FROM identifier phrase 5-28
 - INVALID KEY phrase 5-74
 - mnemonic-names and 3-7
 - modifies LINAGE-COUNTER 4-15
 - organization considerations 5-74
 - ROLLING phrase 7-61
 - STARTING phrase 7-61

Z

zero (0)
 as unique value 4-22
 insertion symbol 4-43
zero filling
 INSPECT statement 5-97
zero suppression and replacement editing 4-52
ZERO, ZEROES, ZEROS figurative constant
 description 2-7
 used as Boolean literal 7-44
zoned decimal item
 description 4-32
 RECORD KEY data item 3-25

Numerics

00-99 segment-numbers, formation rules 2-5
01 level-number
 description 4-18
 illustration 4-20
01-49 level-numbers, formation rules 2-5

02-49 level-number concepts
 description 4-18
 illustration 4-20
1974 Standard COBOL
 description xiv
 1975 FIPS COBOL and 1-3
1975 FIPS COBOL flagging
 full 1-3
 high-intermediate 1-3
 low 1-3
 low-intermediate 1-3
5424 MFCU D-1
66 level-number
 concepts 4-20
 formation rules 2-5
 general description 4-26
 general format 4-24
77 level-number
 concepts 4-20
 formation rules 2-5
88 level-number
 concepts 4-20
 formation rules 2-5
 general description 4-20
 general format 4-24

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Please use this form only to identify publication errors or to request changes in publications.

Possible topics for comment are:

Clarity Accuracy Completeness Organization Retrieval Legibility

If your comment does not need a reply (for example, pointing out a typing error) check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

If you would like a reply, check this box. Be sure to print your name and address below.

Page number (s): Comment(s):

Please print

Name _____

Company or
Organization _____

Address _____

City State Zip Code

No postage necessary if mailed in the U.S.A.

Cut Along Line

IBM System/38 COBOL Reference Manual and Programmer's Guide (File No. S38-24) Printed in U.S.A. SC21-7718-7

Fold and tape

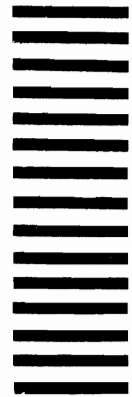
Please do not staple

Fold and tape



NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N. Y.



POSTAGE WILL BE PAID BY . . .

International Business Machines Corporation
Development Laboratory
Information Development, Department 532
Rochester, Minnesota 55901

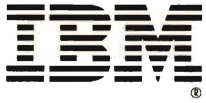
Fold and tape

Please do not staple

Fold and tape



International Business Machines Corporation



International Business Machines Corporation

IBM System/38 COBOL Reference Manual and Programmer's Guide (File No. S38-24) Printed in U.S.A. SC21-7718-7

SC21-7718-07



SC21-7718-7